CygnusCloud: provisión de puestos de laboratorio virtuales bajo demanda

Proyecto de Sistemas Informáticos

Curso 2012/2013

Facultad de Informática
Universidad Complutense de Madrid

Luis Barrios Hernández

Adrián Fernández Hernández

Samuel Guayerbas Martín

Dirigido por

José Luis Vázquez-Poletti

José Antonio Martín Hernández

Nosotros, Luis Barrios Hernández, Adrián Fernández Hernández y Samuel Guayerbas Martín, autores del presente documento y del proyecto *CygnusCloud: provisión de puestos de laboratorio virtuales bajo demanda,* autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Luis Barrios Hernández	**Adrián Fernández Hernández**	**Samuel Guayerbas Martín**
DNI: 53416459-V	**DNI: 05337637-G**	**DNI: 04217069-L**

Agradecimientos

Para realizar este trabajo que sostienes entre tus manos, ha sido imprescindible la colaboración de mucha gente.

En primer lugar, queremos dar las gracias a nuestros directores José Luis Vázquez Poletti y José Antonio Martín Hernández. Sus continuos esfuerzos por dar difusión a nuestro proyecto y su gran entusiasmo siempre nos han empujado a seguir adelante en las horas más bajas.

También queremos dar las gracias a la redacción del programa Hoy por Hoy de la Cadena Ser por habernos concedido el honor de compartir media hora con sus oyentes en riguroso directo. Esta es una experiencia que tardaremos mucho tiempo en olvidar.

Además, también queremos dar las gracias al jurado de la séptima edición del Concurso Universitario de Software Libre por haberse fijado en nuestro humilde proyecto y concederle el premio al mejor proyecto comunitario nacional.

También queremos dar las gracias a nuestras familias: sin el apoyo que nos han demostrado a lo largo de todos estos años, hoy este proyecto no sería una realidad.

Y, por último, no queremos olvidar a los compañeros y amigos que tantas veces nos han soportado y ayudado: gracias por estar ahí.

Resumen

En la actualidad, las aulas de informática de varios centros no disponen de puestos suficientes como para cubrir la demanda. Aunque existen aulas de informática infrautilizadas en el campus, estas no tienen instalado el *software* que utilizan todos los alumnos. Por ello, para realizar trabajos o prácticas, en muchos casos los alumnos deben utilizar sus portátiles o trabajar desde sus casas.

En este documento presentamos un sistema que aprovecha las virtudes de la computación *cloud* para crear puestos de laboratorio virtuales. Estos permitirán que los alumnos de cualquier titulación puedan trabajar desde cualquier aula de informática del campus.

Asimismo, el sistema elimina los lentos procesos burocráticos asociados a la modificación de la configuración de los equipos de las aulas de informática. Al utilizarlo, los profesores podrán modificar una configuración en cuestión de horas (y no de semanas o meses).

Finalmente, la solución propuesta permite reducir los efectos de los recortes en el sector educativo, ya que en las aulas de informática podrán utilizarse equipos más baratos y con un menor consumo energético.

Palabras clave: Cloud Computing, IaaS, virtualización, KVM, libvirt, web2py, twisted, Python.

Abstract

Nowadays, the computers from several faculty labs are often insufficient to meet the demand. Although there are underutilized computer rooms in the campus, the software that every student needs is not always installed on their PCs. Therefore, many students must use their own laptops to do their assignments or work from home.

In this document, we introduce a system that takes advantage of the the cloud computing's virtues to create virtual lab machines. These machines will allow every student from every degree to work at every computer room in the campus.

Furthermore, this system eliminates the time-consuming bureaucratic procedures needed to install or configure software on the computer rooms. By using it, teachers can modify a system configuration within a few hours. Currently, this takes several weeks and even months.

Last, but not least, the proposed solution helps to reduce the effects of budget cuts in education by allowing the use of cheaper and more energy-efficient PCs in the computer rooms.

Keywords: Cloud Computing, IaaS, virtualization, KVM, libvirt, web2py, twisted, Python.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Descripción del problema

1.1. Introducción

El uso de las tecnologías de la información y la comunicación (TIC) en el sector educativo no es algo nuevo. Ya en la década de los 60, Plattner y Herron reconocieron que el uso de computadores era extremadamente útil para enseñar a tomar decisiones en situaciones reales y a desempeñar distintos cargos, y también para fomentar la participación. Por ello, no es de extrañar que las TIC hayan adquirido un papel protagonista en el panorama educativo actual.

De esta manera, a medida que las TIC han ido implantándose en la sociedad, su uso en la universidad no ha hecho más que crecer. Además, la puesta en funcionamiento del llamado Espacio Europeo de Educación Superior ha hecho que las sesiones prácticas tengan un importante peso en todos los planes de estudio universitarios. Y dado que en estas sesiones prácticas se hace uso de las TIC, podemos afirmar que los estudiantes pasarán una cantidad de tiempo considerable delante de un ordenador personal durante su estancia en la universidad.

La Universidad Complutense de Madrid (UCM) es la universidad presencial española con mayor número de alumnos[1]. Para que todos ellos puedan realizar sus trabajos académicos, existe un gran número de aulas de informática y laboratorios repartidos entre sus diversos centros. Naturalmente, la gestión y el mantenimiento de estos espacios es responsabilidad de personal de administración y servicios dependiente, en última instancia, de los Servicios Informáticos de la universidad.

Existen varias alternativas para llevar a cabo la gestión de las aulas de informática. A lo largo de este capítulo, presentaremos la que nos es más conocida y también la que, a causa de sus inconvenientes, ha motivado el desarrollo de *CygnusCloud*: el modelo de gestión de los laboratorios de la Facultad de Informática de la UCM.

A la hora de presentar dicho modelo de gestión, nos centraremos en el *software*, es decir, en la forma en que se instala, configura y se permite utilizar a los estudiantes un conjunto distinto de programas en cada aula de informática. Por ello, en este capítulo no haremos referencia a algunas características también relevantes del modelo como la gestión de altas y bajas de usuarios, el mantenimiento del equipamiento *hardware* del aula y el alojamiento de material docente en la web de la Facultad de Informática.

1.2. El modelo de gestión de los laboratorios de la Facultad de Informática de la UCM

1.2.1. Introducción

Los planes de estudio de la Facultad de Informática de la UCM siempre se han caracterizado por el importante peso asociado a las sesiones prácticas. En el caso de los planes de estudio en extinción, los estudiantes deben pasar más de la mitad de su tiempo de estudio ante un ordenador personal. Además, esta situación se ha mantenido en el caso de los nuevos planes de estudio, implantados en el curso académico 2010/2011.

Así pues, es necesario disponer de espacios en los que los alumnos puedan realizar todas estas actividades prácticas: los *laboratorios*. Para ello, la Facultad de Informática cuenta con 300 ordenadores personales conectados a Internet, repartidos entre 12 laboratorios. En términos generales, podríamos considerar un laboratorio como un aula de informática, si bien algunos disponen de equipamiento específico (como FPGAs, osciloscopios, entrenadores, polímetros, ...).

Con independencia de su equipamiento, del *software* instalado y de las asignaturas que se imparten en los mismos, todos los laboratorios son gestionados por el propio centro.

1.2.2. Requisitos docentes de los laboratorios

En la actualidad, en la Facultad de Informática de la UCM se imparte docencia correspondiente a tres planes de estudio de grado (Grado en Ingeniería de Computadores, Grado en Ingeniería del Software y Grado en Ingeniería Informática), uno de doble grado (Doble Grado en Ingeniería en Informática y Matemáticas), dos ingenierías técnicas (en Informática de Gestión y en Informática de Sistemas) y una ingeniería superior (Ingeniería Informática), junto con múltiples programas de postgrado y doctorado.

Es cierto que la informática es fundamental en todos estos estudios, pero las diferentes asignaturas que se imparten en los laboratorios tienen necesidades muy diferentes. Por ejemplo, una asignatura de introducción a la programación en C++ requerirá un editor y un compilador, mientras que una asignatura de electrónica requerirá osciloscopios, polímetros, resistencias, transistores, ... junto con *software* de simulación de circuitos electrónicos (como PSpice).

Además, en las asignaturas pueden utilizarse herramientas de pago, como MATLAB y *Rational Software Architect*. En la mayoría de casos, los estudiantes no pueden instalar estas aplicaciones en sus equipos por diversos motivos (como el elevadísimo coste de las licencias o la imposibilidad de encontrar la versión instalada en los laboratorios).

De esta manera, la Facultad debe permitir trabajar a los alumnos en los laboratorios fuera del horario de clases no sólo para que utilicen dichas herramientas o el equipamiento específico de estos espacios, sino también para que puedan elaborar los distintos trabajos de clase que se les asignen.

Por último, también es necesario que en los laboratorios puedan tener lugar exámenes de carácter práctico a lo largo de todo el curso académico.

En las siguientes secciones, mostraremos cómo la red de laboratorios satisface todos estos requisitos.

1.2.3. Servicios prestados a los alumnos

Cada alumno matriculado en la Facultad de Informática de la UCM dispone de una cuenta personal que le permite utilizar cualquier puesto de la red de laboratorios. Dicha cuenta tiene asociado un espacio de almacenamiento en disco (100 MB) junto con una cuota de impresión (de 200 hojas por curso académico).

Los servicios de la red de laboratorios que son visibles para los alumnos son los siguientes:

- el control de uso de los puestos de laboratorio. Todos los alumnos deben autenticarse antes de utilizar cualquier puesto, lo que permite detectar y sancionar las actividades indebidas que se realicen en los laboratorios.
- el envío de avisos y notificaciones a los alumnos que utilizan la red de laboratorios.
- el reparto equitativo del tiempo de uso de los puestos de laboratorio en periodos de alta ocupación (restringiendo la duración de las sesiones). Esto garantiza que todos los alumnos puedan utilizar los laboratorios en igualdad de condiciones durante los turnos de práctica libre.
- la consulta de la ocupación en cualquier momento de todos los puestos de la red de laboratorios.

- el almacenamiento de la documentación del material experimental de los laboratorios (como las *datasheets* en las que se recogen las especificaciones de los circuitos integrados que se utilizan en las prácticas).

1.2.4. Servicios prestados a los profesores

La Facultad de Informática de la UCM también ofrece a cada uno de sus profesores una cuenta personal. Esta también les permite utilizar cualquier puesto de la red de laboratorios y dispone de mayores cuotas de almacenamiento y de impresión.

Los profesores también pueden hacer uso de servicios adicionales de apoyo a la docencia. Los más relevantes son los siguientes:

- la restricción de las comunicaciones entre los puestos de laboratorio. Esto permite, entre otras cosas, restringir el acceso a internet durante las clases.
- la recolección de prácticas, exámenes y trabajos realizados por los alumnos. El acceso a los mismos siempre estará limitado al profesor y a su propietario.
- la adquisición, instalación y configuración de todo el *software* que se vaya a utilizar a lo largo del curso académico.

1.2.5. Otros servicios

Junto con los servicios que hemos descrito hasta ahora, que están orientados a los usuarios, existen otros especialmente orientados a facilitar el trabajo del personal técnico, como la monitorización de toda la actividad de los laboratorios en tiempo real y la replicación de configuraciones (que es posible al compartir todos los puestos de un mismo laboratorio la misma configuración *hardware* y *software*).

1.3. Inconvenientes del modelo presentado

1.3.1. Desaprovechamiento de recursos

Uno de los grandes problemas del modelo de gestión que hemos presentado en la sección 1.2 es el desaprovechamiento de recursos.

Los recursos desaprovechados son de naturaleza muy dispar: aulas de informática infrautilizadas que se encuentran en otros centros, PCs de la red de laboratorios, energía, personal, etcétera. Para mostrar cómo se desperdician todos estos recursos, presentaremos algunos ejemplos.

1.3.1.1. Aulas de informática de otros centros

Justo antes de las convocatorias de exámenes y de los periodos vacacionales, los estudiantes deben lidiar con una auténtica “avalancha” de prácticas a entregar en un corto periodo de tiempo. Además, antes de la realización de sus exámenes es bastante frecuente que los alumnos deban revisar las prácticas ya entregadas. Por todo esto, a nadie debe extrañar el elevadísimo uso de los laboratorios en estas épocas del año.

Puesto que la red de laboratorios cuenta con sólo 300 puestos y un número mucho mayor de usuarios (según la memoria de la Facultad del curso 2009/2010, hay más de 2000 alumnos matriculados en este centro), los alumnos deberán competir por los distintos puestos. Ahora bien, no todos los alumnos que utilizan los puestos del mismo laboratorio fuera del horario de clase (horario de práctica libre) tienen por qué estar matriculados en las mismas asignaturas, por lo que necesitarán utilizar *software* (y también posiblemente *hardware*) distinto. Para mostrar cómo se agrava aún más la situación, haremos uso de un ejemplo.

Consideremos la semana anterior a las vacaciones de Navidad. En ella, los alumnos de la asignatura Estructura de Computadores estarán trabajando en una práctica de VHDL de cierta envergadura, mientras que los alumnos de primer curso deberán entregar una práctica de programación en C++.

Supongamos que el simulador y el entorno de desarrollo en VHDL sólo están instalados en los laboratorios 7, 8, 9 y 10. Supongamos también que, en un momento dado, se está impartiendo docencia en los laboratorios 9 y 10. A causa del gran número de alumnos matriculados en la Facultad de Informática, esta situación se da con bastante frecuencia.

Si un alumno desea trabajar en su práctica de VHDL en los laboratorios de la Facultad y fuera del horario de clase (en los denominados turnos de práctica libre), deberá utilizar obligatoriamente los laboratorios 7 y 8. Pero en los turnos de práctica libre estos laboratorios no están reservados en exclusiva para los alumnos de Estructura de Computadores, por lo que los estudiantes de primero también podrán realizar en ellos su práctica de programación.

Si no existen puestos libres en los laboratorios 7 y 8 y un alumno desea continuar con su práctica de VHDL, existen varias alternativas:

- echar de esos laboratorios a los alumnos que no deban utilizar el software específico, es decir, a los alumnos de primero.
- imponer restricciones de tiempo, de modo que los distintos alumnos que utilizan el laboratorio estarán obligados a compartir sus puestos. De esta manera, los alumnos que deseen realizar la práctica en el laboratorio deberán esperar a que haya puestos libres; y los que estén utilizando el laboratorio deberán abandonarlo tras cierto intervalo de tiempo.

Incluso dejando de lado el aumento de la crispación entre los alumnos al que dan lugar ambas alternativas y la situación de auténtico hacinamiento en los laboratorios, esta situación sigue siendo poco deseable: si en un aula de informática de la Facultad de Filología hay puestos libres, estos no podrán utilizarse para completar *cualquiera* de las dos prácticas por el mero hecho de no tener instalado el *software* necesario (ya que las necesidades de los alumnos de esa facultad no justifican la instalación de dicho *software*). En otras palabras: esta situación se puede evitar.

1.3.1.2. Personal encargado de las aulas de informática

Los laboratorios de la Facultad de Informática de la UCM tienen asignado su propio personal de administración y servicios. En periodos de intensa actividad en la Facultad, los técnicos de los laboratorios estarán saturados, mientras que sus compañeros de otros centros pueden tener mucho menos trabajo o incluso estar ociosos.

Esta situación da lugar a mayores tiempos de espera y a un incremento considerable del tiempo de resolución de las incidencias, y mejoraría si los alumnos pudiesen distribuirse de forma más homogénea entre las aulas de informática de todos los centros de la UCM. Nuevamente, esto no es posible al no permitirlo las características técnicas de los PCs y el *software* instalado en las distintas aulas de informática.

1.3.1.3. Energía consumida

Hoy en día, el coste de la energía no es despreciable en absoluto. Prueba de ello es que los centros de proceso de datos de más reciente construcción se caracterizan por la búsqueda de la mayor eficiencia energética posible para así reducir lo máximo posible los costes de explotación.

Ante la actual situación de crisis económica, la Universidad Complutense no puede quedarse atrás en la mejora de la eficiencia energética: el ahorro energético permite reducir el efecto que las sucesivas reducciones del presupuesto tienen sobre otras partidas (como las dedicadas a becas y a investigación). Por ello, es fundamental racionalizar el consumo energético de las aulas de informática.

La primera fuente de consumo innecesario es el sistema de climatización y alumbrado de las aulas de informática infrautilizadas, cuyo uso resulta imprescindible con independencia del número de alumnos que estén trabajando en el aula.

La segunda fuente de consumo es el propio equipamiento informático asignado a las aulas infrautilizadas: los PCs, servidores y también el equipamiento de red. Aún apagando los equipos en desuso, se sigue desperdiciando energía, ya que

- frecuentemente, la red no se utiliza para más que para acceder a internet o a la intranet de la UCM. Estas actividades utilizan muy poco ancho de banda, por lo que el equipamiento de red y los servidores asignados al aula de informática están siendo frecuentemente infrautilizados.
- los equipos en uso están inactivos durante buena parte de su vida útil, y el consumo energético en periodos de inactividad no es despreciable.

Estos consumos pueden reducirse si los alumnos que utilizan estas aulas se agrupan en un único aula de informática, cosa que no permite el modelo actual por motivos que ya hemos expuesto:

- los PCs del aula de informática pueden no ser lo suficientemente potentes como para cubrir las necesidades de todos los estudiantes, y
- los alumnos de distintas titulaciones no tienen por qué utilizar el mismo *software*.

1.3.1.4. Licencias de pago

En las aulas de informática de la UCM se encuentran instaladas aplicaciones de pago. Puesto que las licencias de dichas aplicaciones son muy costosas, la UCM garantiza que todos sus alumnos puedan utilizarlas en las aulas de informática.

Ahora bien, tal y como hemos expuesto en las secciones anteriores, no todos los alumnos utilizan las aulas de informática para lo mismo. Así, a causa de la mera ocupación de ciertos espacios, no es posible utilizar todas las licencias de pago adquiridas por la UCM aún cuando es necesario. También podemos aprovechar el ejemplo que presentamos en la sección 1.3.1.1 para mostrar cómo esto puede ocurrir.

Los alumnos que deseen avanzar con su práctica de VHDL necesitan utilizar un entorno de desarrollo de VHDL y un simulador. Este *software* estará instalado en todos los puestos de los laboratorios 7 y 8, pero no se utilizará en los puestos ocupados por los alumnos de primero. De esta manera, aunque las licencias de estos puestos no se utilizan, no será posible aprovecharlas al no estar instalado el *software* en otras aulas de informática.

Además, puesto que también la adquisición de *software* se realiza a través de la red de laboratorios, resulta imposible compartir las licencias sin utilizar con otros centros, lo que incrementa considerablemente el gasto en *software*. Por ejemplo, si tanto en la Facultad de Ciencias Físicas como en la Facultad de Informática se utiliza el *software* MATLAB, estos dos centros no podrán compartir sus licencias de MATLAB porque las aulas de informática de Físicas no forman parte de la Red de Laboratorios de la Facultad de Informática.

Considerando el desorbitado precio de las aplicaciones propietarias en general, las limitaciones de las versiones de prueba de muchas de ellas y la imposibilidad de encontrar en el mercado las versiones instaladas de algunas de las que se utilizan para realizar prácticas (frecuentemente, por ser demasiado antiguas), esta situación resulta inadmisible.

1.3.2. Limitaciones del sistema de almacenamiento

Toda cuenta de la red de laboratorios de la Facultad de Informática tiene asignado un espacio de almacenamiento en disco accesible desde cualquier puesto.

Ahora bien, la cuota actual asignada a los alumnos (100 MB) es insignificante, y en no pocos casos es sólo suficiente para almacenar el material docente de las distintas asignaturas.

Por otra parte, sólo es posible acceder a ese espacio de almacenamiento desde la propia red de laboratorios, siendo necesario utilizar dispositivos de almacenamiento extraíble o servicios de alojamiento de archivos en la nube (como *Dropbox*, *Microsoft SkyDrive* y *Google Drive*) para transferir datos entre los equipos particulares de los estudiantes y los PCs de la red de laboratorios.

1.3.3. Sobrecoste de los PCs de las aulas de informática

En el modelo de gestión de los laboratorios de la Facultad de Informática de la UCM no existen laboratorios asignados en exclusiva a una asignatura o departamento. Por ello, en un mismo laboratorio se imparte docencia de asignaturas distintas.

Ahora bien, no todas las asignaturas asignadas al mismo laboratorio tienen las mismas necesidades. Por ello, los PCs de cada laboratorio deben ser lo suficientemente potentes para así cumplir los requisitos del *software* que utilizan dichas asignaturas.

Para cuantificar el sobrecoste de los equipos, consideraremos tres asignaturas típicas: una asignatura de programación, una asignatura relacionada con la informática gráfica y una asignatura de diseño de circuitos. Actualmente, estos tipos de asignatura se imparten en los laboratorios de la Facultad de Informática.

- La asignatura de programación requiere un PC con una CPU de gama media-baja, al menos 4 GB de memoria RAM, una tarjeta gráfica integrada y un disco duro de capacidad estándar.
- La asignatura de informática gráfica requiere un PC con una CPU de gama media-alta, al menos 4 GB de memoria RAM, una tarjeta gráfica dedicada de gama media-baja y un disco duro de capacidad estándar.
- La asignatura de diseño de circuitos requiere un PC con una CPU de gama media-alta, 4 GB o más de memoria RAM, una tarjeta gráfica integrada y un disco duro de capacidad estándar.

Para realizar la comparación, hemos escogido un proveedor al azar, al que llamaremos Proveedor A. A fecha de 25 de octubre de 2012, los precios y características de los equipos requeridos por las tres asignaturas, a los que llamaremos equipos 1, 2 y 3, son los siguientes:

- Equipo 1 (programación): *Asustek* CM6431, con CPU Intel Core i3 3220, 4 GB de memoria RAM, tarjeta gráfica integrada Intel HD Graphics 3000, grabadora de DVDs y disco duro de 1 TB. Su coste es de 784,30 €, con IVA incluido.
- Equipo 2 (informática gráfica): HP P6-2206ES, con CPU Intel Core i5 2320, 8 GB de memoria RAM, tarjeta gráfica Nvidia Geforce GT 620, grabadora de DVDs y disco duro de 1 TB. Su coste es de 960,16 €, con IVA incluido.
- Equipo 3 (diseño de circuitos): *Asustek* CM6431, con CPU Intel Core i5 3450, 6 GB de memoria RAM, tarjeta gráfica integrada Intel HD Graphics 3000, grabadora de DVDs y disco duro de 1 TB. Su coste es de 941,48 €, con IVA incluido.

Para impartir las tres asignaturas en el mismo laboratorio, sería necesario adquirir el Equipo 2. Dado que el laboratorio tiene 20 puestos,

- si solamente se impartiera la asignatura de programación, el ahorro sería de

$$20 \cdot (960,16 - 784,30) = 20 \cdot 175,86 = 3517,2 \text{ €}$$

- si solamente se impartieran las asignaturas de programación y diseño de circuitos, el ahorro sería de

$$20 \cdot (960,16 - 941,48) = 20 \cdot 18,64 = 373,6 \text{ €}$$

En realidad, el sobrecoste es aún mayor por dos motivos:

- cuanto más potente es el *hardware* de un PC, mayor es su consumo energético. De esta manera, cuando se imparte la asignatura de programación la CPU y la tarjeta gráfica del Equipo 2 consumen más energía de la requerida. Análogamente, cuando se imparte la asignatura de informática gráfica, será la tarjeta gráfica la que eleve el consumo innecesariamente.
- las licencias del sistema operativo Windows incluidas con el equipo no se utilizan (la UCM utiliza sus propias licencias). El coste de estas licencias, llamadas licencias OEM (*Original Equipment Manufacturer*, fabricante de equipamiento original) es de 102,90 € (IVA incluido) para las versiones *Home Premium* y de 142,90 € (IVA incluido) para las versiones *Professional*.

  De esta manera, en el caso mejor, en el que todos los equipos incorporan la versión más barata, *Home Premium*, el sobrecoste aumenta en

$$20 \cdot 102,90 = 2058 \text{ €}$$

y en el caso peor, en el que todos los equipos incorporan la versión más cara (*Professional*), el sobrecoste aumenta en

$$20 \cdot 142,90 = 2858 \text{ €}$$

1.3.4. Excesiva rigidez de la configuración de los equipos

Tal y como hemos descrito en la sección 1.2.4, entre los servicios que presta la red de laboratorios de la Facultad de Informática de la UCM se encuentra la adquisición, instalación y configuración de todo el *software* que se vaya a utilizar a lo largo del curso académico.

Este servicio evita a los docentes la tediosa labor de configuración de todos los equipos de los laboratorios, permitiendo que estos se dediquen en mayor medida a la atención del resto de sus obligaciones.

Pero también tiene un serio inconveniente: la instalación de nuevo *software* y la modificación de la configuración de los equipos de los laboratorios se convierte en una auténtica "carrera de obstáculos" burocrática, que da lugar a ingentes retrasos que afectan muy negativamente al normal desarrollo de las clases en los laboratorios.

Para ilustrar por qué ocurre esto último, describiremos el procedimiento a seguir para fijar la configuración de los equipos de los laboratorios que, en el presente curso académico (2012/2013), es el siguiente:

1. A principio de curso, los profesores escogen de entre todos los programas que figuran en una lista aquellos que necesitarán utilizar. Estos no tienen por qué instalarse en todos los laboratorios: cada profesor sólo puede solicitar la instalación de *software* en los laboratorios que utilizará en sus clases.

2. Los técnicos proceden a realizar la instalación de dicho *software* en las máquinas de prototipado de los distintos laboratorios.

3. Se crean imágenes de disco de las máquinas de prototipado, que se utilizarán para replicar la configuración de las mismas en todos los puestos de laboratorio.

4. En la primera semana de cada cuatrimestre, los profesores verifican la configuración de los laboratorios que utilizarán en sus clases. Durante este periodo de pruebas, los alumnos no podrán utilizarlos.

5. Los laboratorios se abren a todos los alumnos para su uso en turnos de prácticas a partir de la segunda semana de cada cuatrimestre.

Lo primero que llama la atención es el hecho de que *todos* los profesores de *todas* las asignaturas que se imparten en los laboratorios tienen *una* semana para verificar la configuración de los equipos. Teniendo en cuenta el gran número de titulaciones que se imparten en la Facultad de Informática, esta restricción impone un ritmo frenético a la hora de comprobar la correcta configuración de los equipos, lo que facilita que muchos errores se pasen por alto.

Es más, en caso de que se detecte alguna anomalía, esta deberá subsanarse durante esa misma semana, lo cual propicia la aparición de muchos más errores. Así, es bastante habitual que no todo el *software* instalado en los puestos de laboratorio funcione de forma correcta, siendo fácil encontrar aplicaciones en las que no es posible utilizar ni la funcionalidad más básica.

Además, es importante notar que los profesores sólo pueden escoger el *software* a instalar de una lista. Si un programa no aparece en dicha lista, deberán solicitar su instalación al Analista, lo cual da lugar a otra sucesión de trabas burocráticas. En caso de aplicaciones de pago, cuyas licencias suponen un coste importante para la Facultad, esto es bastante razonable. Pero esta situación resulta injustificable cuando se trata de *software* gratuito o que no supondrá ningún sobrecoste para la Facultad, y da lugar a retrasos que, de acuerdo a nuestra experiencia, superan fácilmente el mes de duración.

Pero el problema no acaba aquí: si un profesor detecta un error en la configuración de los puestos tras la semana de pruebas, también será necesario iniciar otro lento proceso burocrático para algo

tan sumamente simple como volver a ejecutar algunos instaladores en la máquina de prototipado, volver a generar la imagen de disco y clonarla.

En definitiva, el modelo actual da lugar a una sobreacumulación de retrasos y errores en la configuración de los equipos que degrada considerablemente la calidad de la docencia e impide a los alumnos el uso normal de los puestos de laboratorio, obligándoles en no pocos casos a utilizar sus propios equipos durante una parte muy significativa del curso académico.

Capítulo 2

Cloud Computing

2.1. Introducción

El *Cloud Computing* o computación en la nube es el resultado de la evolución de los modelos de cómputo existentes, y permite al cliente abstraerse del *hardware* con el cual va a trabajar y centrarse en los servicios que necesita para realizar dicho trabajo. De esta forma, el cliente solo debe preocuparse por lo que quiere hacer y no por configurar el entorno que necesita.

Así, en este nuevo modelo pueden combinarse muchas clases de recursos en tiempo real para satisfacer las especificaciones o proporcionar los servicios que el cliente necesita. Estos recursos pueden ser de naturaleza muy distinta: tiempo de CPU, espacio de almacenamiento, infraestructura de comunicaciones, etcétera.

Además, el *Cloud Computing* elimina los costes de adquisición del *hardware*, las licencias del software, el mantenimiento de las instalaciones, etcétera.

FIGURA 2.1: Computación en la nube: un servicio global

En este capítulo abordaremos el estudio de *Cloud Computing*, que proporciona la sólida base sobre la que hemos desarrollado *CygnusCloud*. Para ello, comenzaremos definiéndolo detalladamente para después explicar su evolución y finalizar describiendo las características y los diferentes tipos de sistemas en la nube.

2.2. Definición

Según indica el Instituto Nacional de Estándares y Tecnologías de Estados Unidos (NIST) en el documento [2], el *Cloud Computing* es

> *un modelo que permite acceder a través de una red, de forma adecuada, desde cualquier sitio y bajo demanda a un conjunto compartido de recursos informáticos configurables (como*

> *redes, servidores, almacenamiento, aplicaciones, servicios. . .), que se pueden proporcionar y utilizar rápidamente y con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios.*

En esta definición podemos distinguir ya las cuatro características fundamentales del *Cloud Computing*:

- el autoservicio o administración propia de los servicios ofertados,
- la accesibilidad desde múltiples plataformas,
- la compartición de los recursos entre varios usuarios, y
- la prestación de servicios flexibles y perfectamente ajustables al entorno de trabajo.

A continuación, mostraremos los precedentes y la evolución histórica del *Cloud Computing*. Más adelante, en la sección 2.4, hablaremos detalladamente de estas características.

2.3. Historia

2.3.1. Precedentes

Aunque las ideas que hay detrás del *Cloud Computing* pueden parecer novedosas, en realidad están presentes en nuestra sociedad desde hace siglos. Para justificar esto, utilizaremos un ejemplo.

Con independencia de nuestro lugar de procedencia, edad o condición, todos nosotros necesitamos una misma cosa para subsistir: agua. Así, los primeros humanos ya se asentaban en territorios cercanos a ríos y lagos para poder acceder al agua dulce.

Siglos después, el crecimiento de la población hizo necesaria la construcción de pozos para extraer el agua subterránea. Según los registros históricos, los primeros pozos se excavaron en Jericó (Israel) hace unos 7000 años.

La concentración de la población dio lugar a la aparición de los primeros sistemas de almacenamiento y distribución de agua potable. Los primeros que fueron lo suficientemente eficientes y que transportaron agua con la calidad suficiente aparecieron en la antigua Grecia, y los primeros que lograron cubrir grandes distancias, los famosos acueductos, se construyeron durante el Imperio Romano.

Tras el retroceso que experimentaron durante la Edad Media, los sistemas de distribución de agua volvieron a adquirir protagonismo en la Edad Contemporánea. Así, el primer sistema de suministro capaz de abastecer (mediante fuentes) a una ciudad completa (Paisley, Escocia) fue diseñado por John Gibb a principios del siglo XIX.

Hoy en día, todos nosotros disponemos de agua potable purificada abriendo cualquier grifo de nuestra casa. Cada vez que abrimos un grifo, estamos obviando

- la procedencia del agua,
- los embalses, pozos o depósitos en los que se almacena,
- la planta de tratamiento en la que se ha purificado,
- el diseño de las redes de distribución y de alcantarillado, y
- la planta donde se depurarán las aguas residuales que generamos en nuestro hogar.

De esta manera, aunque el sistema de distribución de agua puede llegar a ser muy complejo, para disponer de agua potable sólo debemos preocuparnos por pagar las facturas de la compañía suministradora.

Esta es la misma idea que hay detrás del *Cloud Computing*: los usuarios se preocupan únicamente por utilizar los servicios que han contratado y no por la forma en que se los presta su proveedor.

2.3.2. Aparición de las ideas básicas del *Cloud Computing*

El origen de las ideas básicas que han dado lugar al *Cloud Computing* se remonta a la década de los 60.

Así, en 1961 John McCarthy dio una gran importancia al hecho de que el mundo de la informática debía ser de uso público. Por otra parte, J.C.R Licklider introdujo en 1962 la idea de una "red de ordenadores intergaláctica", que interconectaría a todo el mundo a lo largo y ancho del planeta y haría posible la compartición de información, datos y programas.

2.3.3. Primeros años de los computadores en la empresa

En la década de los 80, los sistemas informáticos de las empresas operaban de acuerdo a un modelo cliente/servidor: todas las aplicaciones y todos los datos estaban almacenados en enormes computadores, los servidores u ordenadores centrales. Así, si un usuario quería acceder a cierta información o ejecutar un determinado programa, tenía que conectarse al ordenador central a través de un terminal o estación de servicio.

Pero los ordenadores centrales tenían unos recursos limitados, el acceso a los mismos no era inmediato y no soportaban la concurrencia, es decir, dos usuarios no podían acceder a la misma información a la vez. Por ello, los usuarios debían esperar a que el ordenador central atendiese sus peticiones, lo que daba lugar a enormes retardos.

2.3.4. Aparición del modelo *peer to peer*

El gran inconveniente del modelo cliente/servidor es que el servidor se convierte en un cuello de botella, ya que todos los usuarios necesitan acceder al servidor para trabajar. El modelo *peer to peer* (P2P) surgió a partir de la idea de no tener que utilizar el servidor como intermediario a la hora de establecer una comunicación entre dos ordenadores. En este modelo, todos los ordenadores son a clientes y servidores a la vez.

La principal finalidad con las que se desarrollo el modelo P2P fue su uso en internet: los mensajes podían propagarse desde cualquier ordenador, lo que permitía incrementar la velocidad de trasferencia al no producirse las transmisiones desde un único punto.

En la práctica, puesto que el procesamiento de información requiere tiempo y energía y no todos los ordenadores están conectados permanentemente, la Red no sigue este modelo de forma estricta: cada usuario sólo puede conectarse a un conjunto de servidores, lo que le proporciona mayor libertad con respecto al modelo cliente/servidor y permite restringir el control y la gestión de los datos a un conjunto de servidores centrales.

2.3.5. Desarrollo del *Cloud Computing*

A finales de la década de los 90 aparecieron las primeras empresas que empezaron a proporcionar servicios a otras a través de la *web* y con un coste considerablemente inferior. Entre todas ellas, destaca *Salesforce.com*, que comenzó a vender aplicaciones a empresas a través de la *web* en 1999.

Tras el estallido de la burbuja .com, *Amazon* pasó a jugar un papel protagonista en el desarrollo del *Cloud Computing* al iniciar la modernización de sus centros de proceso de datos. Para evitar su infrautilización, *Amazon* apostó por la nueva arquitectura *Cloud* y lanzó al mercado el paquete *Amazon Web Services* en julio de 2002. Este incluía almacenamiento, capacidad de cómputo e incluso aprovechaba la inteligencia humana (en el caso del servicio *Amazon Mechanical Turk*).

Cuatro años más tarde, en 2006, *Amazon* lanzó al mercado su producto *Elastic Compute Cloud*, que constituye una infraestructura completa y accesible de *Cloud Computing* y que permite alquilar máquinas virtuales en las que individuos y pequeñas empresas pueden alojar sus propias aplicaciones.

En 2009, Google dio el gran salto al lanzar *Google Apps*, un conjunto de aplicaciones diseñadas para ser utilizadas a través de un navegador web.

Por otra parte, las *killer apps* creadas por empresas como *Microsoft* y *Google* también han tenido una gran importancia en el desarrollo del *Cloud Computing*. Un claro ejemplo son los servicios de

correo web, que han desterrado a los clientes de correo electrónico tradicionales en muchos casos y han contribuido a que todos los usuarios estén utilizando tecnologías *Cloud* en su vida diaria.

En la actualidad, resulta evidente que el *Cloud Computing* adquiere cada vez más importancia y que ha venido para quedarse: hoy en día, es posible acceder a estos servicios desde cualquier plataforma y desde casi cualquier lugar del mundo, lo que hace que cada vez más aplicaciones se adapten al *Cloud*.

Además, la mejora de la seguridad permite que las empresas puedan ampliar su infraestructura o externalizarla, lo que se traduce en un aumento de la flexibilidad, en un mayor número de servicios disponibles y en un importante ahorro de coste.

FIGURA 2.2: El *Cloud Computing* en la actualidad

2.4. Características

Las características que han permitido que el *Cloud Computing* sea un servicio en auge en la actualidad son las siguientes:

- **Agilidad**. El usuario puede gestionar de manera sencilla las aplicaciones que necesita en su entorno de trabajo. Así, el usuario puede adquirir o dejar de utilizar sus aplicaciones según sus necesidades y de manera fácil y rápida.

- **Reducción de costes**. Dado que existen empresas dedicadas a alquilar *hardware* a los usuarios interesados en ejecutar sus aplicaciones en la nube, estos no necesitarán hacer frente a los gastos iniciales de compra de recursos ni a los gastos de mantenimiento.

  Si bien es cierto que es necesario pagar un alquiler por los servidores utilizados, este es menor a los posibles gastos a corto y largo plazo a los que habría que hacer frente.

- **Accesibilidad**. La independencia entre el entorno de trabajo y la localización del servidor donde se ejecuta el servicio permite al usuario acceder a sus herramientas de trabajo desde cualquier dispositivo con acceso a la web.

  De esta forma, los clientes pueden trabajar desde sus equipos de sobremesa, sus portátiles o incluso desde sus dispositivos móviles.

- **Elasticidad** y **escalabilidad**. Las aplicaciones en la nube son totalmente elásticas en cuanto a su sencillez de implementación y adaptabilidad. Además, son totalmente escalables. Por ejemplo, el sistema de una empresa puede modificarse para a atender al triple de usuarios con total normalidad y rapidez.
- **Recuperación**. Los proveedores de servicios en la nube proporcionan a los usuarios sistemas de almacenamiento secundario en servidores. Así, si se producen pérdidas de información estas pueden resolverse de manera inmediata.
- **Estabilidad**. Las empresas de gestión de las aplicaciones en la nube aseguran al usuario una fuerte estabilidad en su entorno y permiten un rápido restablecimiento de los servicios que experimenten algún problema.
- **Seguridad**. Quizás esta es la característica que suscita una mayor controversia el *Cloud Computing*. Los clientes de estos servicios pueden tener la sensación de que al almacenar su información en un lugar externo a su propio dispositivo, resulta más sencillo que terceras personas puedan acceder a la misma.

  Sin embargo, esto no es así en realidad. Las empresas encargadas de ofertar servicios en la nube mantienen estrictos controles de seguridad para evitar que la información de sus clientes pueda ser filtrada al exterior.

  Además, para almacenar esta información suelen utilizar centros de datos secundarios especializados en la protección y vigilancia de datos.

2.5. Modelos de servicio

El *Cloud Computing* ofrece tres modelos de servicio fundamentales, que distinguen por la forma en que los clientes utilizan sus servicios contratados en la nube.

Estos modelos no son siempre excluyentes entre sí, y existen proveedores que mezclan algunos de ellos. Por ejemplo, el servicio *Google App Engine* permite que los clientes ejecuten aplicaciones web sobre la infraestructura de *Google*. Así, se están combinando el modelo de plataforma como servicio de la infraestructura web de *Google* y un modelo de *software* como servicio.

2.5.1. Infraestructura como servicio (IaaS)

Su nombre procede del termino inglés *Infrastructure as a Service* (IaaS). Este es el modelo más básico. En él, el proveedor oferta al usuario un conjunto de maquinas virtuales sobre las que trabajar. Así, el cliente estará contratando únicamente la infraestructura tecnológica, que dispone de cierta capacidad de procesamiento, de almacenamiento y de comunicación.

Cuando el cliente contrata la infraestructura y el proveedor acaba de configurarla, el cliente la utilizará para alojar sus aplicaciones, que se ejecutarán en máquinas virtuales. Estas maquinas virtuales pueden gestionarse a través de hipervisores (como Xen o KVM) o a través de otros sistemas de virtualización como *VirtualBox* o *VMWare*.

Puesto que el proveedor del servicio se ocupará de la gestión de las máquinas virtuales, el cliente puede contratar tantas maquinas virtuales como necesite.

En cualquier caso, una vez obtenidas las maquinas virtuales el cliente instalará en ellas el sistema operativo que más le interese y las aplicaciones que vaya a utilizar. Por ello, en este modelo el cliente es el responsable del mantenimiento y configuración del entorno.

Además, puesto que los proveedores no deben preocuparse por otros aspectos como la seguridad o el pago de licencias, únicamente cobran por la cantidad de recursos asignados y consumidos.

Algunos servicios de este tipo son *Amazon CloudFormation*, *Rackspace Cloud*, *Terremark*, *Windows Azure Virtual Machines* y *Google Compute Engine*.[3]

2.5.2. Plataforma como servicio (PaaS)

Su nombre procede del termino inglés *Platform as a Service* (PaaS). Los proveedores de este tipo de servicios ofrecen a sus clientes una plataforma completa, la cual incluye el sistema operativo,

el conjunto de todas las aplicaciones instaladas, la gestión de las bases de datos y el conjunto de servicios *web* que el cliente necesite. Esta plataforma se instalará sobre una infraestructura, también proporcionada por el proveedor.

De esta forma, en este caso el cliente despliega las aplicaciones que necesite (que pueden estar desarrolladas por él mismo o por terceros) sin preocuparse por los costes de compra y de mantenimiento del *hardware* y del *software*.

Este modelo resulta ventajoso tanto para el cliente como para el proveedor, ya que

- el primero no necesita costear el precio completo de las licencias de todas las herramientas que utiliza. Esto resulta especialmente interesante en el caso de *software* muy caro y muy poco utilizado por el cliente: en estos casos, la adquisición de una licencia completa es un despilfarro.
- el segundo puede ofrecer un mismo entorno a un gran número de clientes con necesidades similares.

Algunos servicios de este tipo son *Amazon Elastic Beanstalk, Cloud Foundry, Heroku, Force.com, EngineYard, Mendix, Google App Engine, Windows Azure Compute y OrangeScape*.

2.5.3. Software como servicio (SaaS)

Su nombre procede del termino inglés *Software as a Service* (Saas). Este modelo es el que proporciona menos libertad (y también menos preocupaciones) al cliente. En este caso, el proveedor no solo ofrece la infraestructura y las herramientas, sino que sobre ellas instala el conjunto de aplicaciones que utilizará el cliente.

Esto libera al cliente de instalar y, ejecutar las aplicaciones, y también de realizar las labores de mantenimiento, recuperación y soporte de las mismas.

El coste de alquiler de las aplicaciones en un modelo de software como servicio suele estar ligado a una tarifa plana anual[4] o mensual por usuario[5]. Este precio dependerá del número de usuarios que vayan a usar la aplicación y puede reajustarse a medida que se realicen las altas y bajas de usuarios.

Algunos ejemplos de software como servicio son *Google Apps, innkeypos, Quickbooks Online, Successfactors Bizx, Limelight Video Platform, Salesforce.com y Microsoft Office 365*.

2.6. Otros modelos de servicio especiales

Aparte de los tres modelos de servicio fundamentales, existen otros modelos menos relevantes que han surgido para resolver problemas muy puntuales. En general, la mayoría de estos modelos secundarios han surgido mezclando dos los modelos fundamentales.

2.6.1. Escritorio como servicio (DaaS)

Su nombre procede del término inglés *Desktop as a Service* (DaaS). En el caso de este servicio, también conocido como escritorio virtual, se ofrece al cliente un escritorio con todas las aplicaciones convenientemente instaladas y listas para ser ejecutadas.

Varios usuarios pueden trabajar en paralelo sobre el mismo escritorio virtual sin que las acciones de unos afecten al resto.

Así, cuando el usuario inicia sesión podrá utilizar su configuración personal y toda la información que previamente había guardado en el escritorio. Además, cuando el usuario cierra la sesión se almacenarán la configuración y los datos que haya salvado.

2.6.2. Entorno de pruebas como servicio (TEaaS)

Su nombre procede del término inglés *Test Environment as a Service* (TEaaS). Este modelo ofrece al cliente un entorno en el que probar un determinado software.

2.7. Tipos de *cloud*

En función de la ubicación de los servidores y del público al que están destinados los servicios, podemos distinguir cuatro tipos de *cloud*: *cloud* público, *cloud* privado, *cloud* híbrido y *cloud* comunitario.

2.7.1. *Cloud* público

Es el tipo más común, y también el que mejor representa la idea de computación en la nube. En caso, los servicios están destinados al público en general y suelen ser gratuitos o estar ligados a un modelo de pago por uso.

El uso de *clouds* públicos resulta muy adecuado cuando hay que

- controlar los picos de carga.
- desarrollar una aplicación utilizando un modelo de plataforma como servicio (PaaS)[6].
- utilizar un modelo de software como servicio (SaaS) con una buena estrategia de seguridad.
- desplegar aplicaciones que son utilizadas por muchos usuarios y que, de otra manera, requerirían una gran inversión en infraestructuras.
- utilizar una capacidad de almacenamiento incremental. En estos casos, la capacidad que va a necesitar no se puede estimar.

Las principales ventajas que ofrece el uso de *clouds* públicos son la gran flexibilidad, la seguridad y el ahorro en la infraestructura. Además, no requieren grandes inversiones, garantizan la escalabilidad y no tienen límite de recursos ni de disponibilidad.

Algunos *clouds* públicos son el *Elastic Compute Cloud* de *Amazon* y *Windows Azure* de *Microsoft*[2, 7].

2.7.2. *Cloud* privado

Es el segundo tipo de computación en la nube más utilizado. En general, se acepta que cualquier centro de datos propiedad de una organización, destinado a su uso exclusivo y con cierto nivel de virtualización y autonomía es un *cloud* privado.

Los proveedores de *clouds* privados suelen ofrecer centros de procesamiento de datos a grandes empresas. Por ello, en estas instalaciones debe prestarse una mayor atención a la seguridad y a la integridad de los datos que se manejan, lo que hace que los *clouds* privados resulten algo más costosos que el resto.

Las principales ventajas que ofrecen los *clouds* privados son el aislamiento y la exclusividad de su propia infraestructura, así como máximas garantías de seguridad y privacidad en sus sistemas de procesamiento de información [2, 7, 8, 9].

Finalmente, el uso de *clouds* privados resulta muy adecuado cuando

- el negocio del cliente se basa en su información y sus aplicaciones. Por lo tanto, el control y la seguridad son primordiales.
- el negocio del cliente está relacionado con la gestión de la información de otras empresas.
- una organización es lo suficientemente grande como para poder mantener un sistema en la nube propio y eficiente[6].

2.7.3. *Cloud* híbrido

Los *clouds* híbridos mezclan las características más importantes de los *clouds* privado y de los *clouds* públicos, de manera que

- el cliente gestiona y explota en exclusividad su infraestructura, y
- dispone de acceso al *cloud* público que mantiene el proveedor en sus instalaciones.

Podemos distinguir dos tipos de *clouds* públicos:

- *cloud* privado *in-house*. En este caso, la infraestructura se encuentra físicamente en las instalaciones del cliente. Este modelo requiere una importante inversión económica y de recursos por parte del cliente. Además, la seguridad de la información depende de su experiencia y de la capacidad técnica de la que disponga.

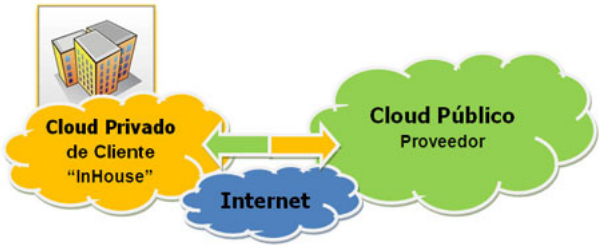

FIGURA 2.3: Cloud privado *in-house*

- *cloud* privado y *cloud* público en *datacenter* del proveedor. En este caso, la infraestructura del cliente puede residir en el centro de datos del proveedor o incluso puede subcontratarse, lo que hace desaparecer la necesidad de una inversión inicial.

  Esto permite reducir costes mediante un modelo de pago por uso y garantizar la escalabilidad de forma rápida, flexible y sin necesidad de una inversión extra [2, 10, 7].

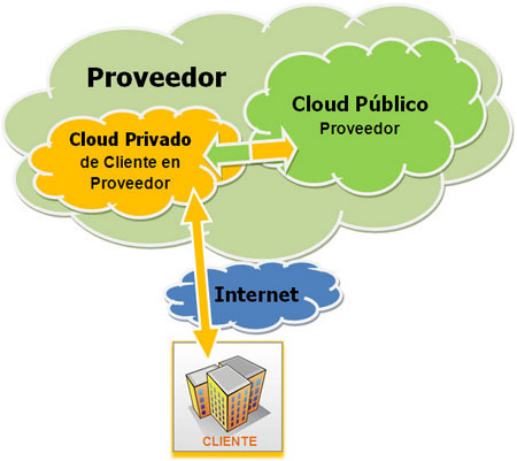

FIGURA 2.4: Cloud híbrido en el centro de proceso de datos del proveedor

Finalmente, el uso de un *cloud* híbrido resulta muy adecuado cuando una organización

- utiliza una infraestructura privada para realizar sus actividades y delega en servicios públicos para realizar el almacenaje de la información.
- utiliza un modelo de software como servicio y necesita una infraestructura privada para almacenar información con unos requisitos de seguridad muy elevados[6].

2.7.4. *Cloud* comunitario

En este caso, dos o más organizaciones forman una alianza y comparten una infraestructura con un marco de seguridad y privacidad común. Esto les permite beneficiarse de una exclusividad casi total reduciendo el coste de compra y de mantenimiento de servidores.

El principal inconveniente de este modelo es la pérdida del nivel de seguridad característico de los *clouds* privados, ya que esta pasa a depender de la organización que albergue la infraestructura [2, 7].

2.8. Ventajas e inconvenientes

2.8.1. Ventajas

Las principales ventajas del *Cloud Computing* son las siguientes:

- **Accesibilidad**. El uso del *Cloud Computing* permite que todos los empleados puedan acceder a los datos de la organización de forma fácil y rápida con independencia del lugar en el que se encuentren.
- **Reducción de costes en infraestructura**, **mantenimiento y servicios**. Al utilizar estas tecnologías, se reducen los gastos de adquisición y mantenimiento de los servidores sobre los que se ejecutan las aplicaciones.
- **Escalabilidad**. Las infraestructuras *cloud* son capaces de suministrar recursos, *software* y/o datos en función de la demanda existente sin que los usuarios perciban diferencias.
- **Seguridad** para mantener la integridad de los datos. Esta dependerá del *cloud* que se utilice.
- **Fácil integración de servicios**. Las aplicaciones en la nube pueden integrarse en con otras de forma rápida y sencilla.
- **Prestación de servicios a nivel mundial**. Con independencia de la ubicación de la infraestructura, los servicios que prestan los proveedores pueden utilizarse en cualquier parte del mundo de manera rápida y sencilla.
- **Actualizaciones automáticas**. En los sistemas convencionales, cada vez que se realiza una actualización de la aplicación el cliente debe dedicar tiempo y esfuerzo para volver a personalizarla e integrarla. Con la computación en la nube, estas actualizaciones se realizan de manera automática y sin que el cliente se vea afectado.

2.8.2. Desventajas

El *Cloud Computing* también tiene varios inconvenientes. Los principales son los siguientes:

- **Privacidad**. Esta es más una cuestión de confianza de los usuarios que un problema de la computación en la nube en sí. Al estar almacenados los datos de los usuarios en equipos externos, estos temen que otros puedan acceder fácilmente a su información.
- La **gran dependencia del proveedor de servicios en la nube**. Cuando un cliente opta por no utilizar un *cloud* privado, pasa a depender de su proveedor para desarrollar total o parcialmente su actividad [11].
- La disponibilidad de los servicios está estrechamente ligada al correcto funcionamiento de la red.
- En la actualidad existen ciertas necesidades que aún la nube no puede cubrir.
- **La escalabilidad debe planificarse con cuidado**. A medida que más usuarios empiecen a compartir misma infraestructura, los servidores se encontrarán más sobrecargados. Si la organización no ha hecho las previsiones oportunas, la calidad del servicio puede degradarse de forma muy importante.

2.9. El *Cloud Computing* como un estándar abierto

Si tenemos en cuenta el desarrollo de la computación en nube hasta la fecha, podemos observar que esta tecnología es el resultado de la convergencia de muchas normas diferentes.

El crecimiento del *Cloud Computing* ha dado lugar a un importante impulso por parte de la industria para crear estándares abiertos basados en la nube. Así, en la actualidad, el *Cloud Computing* se rige por los siguientes estándares tecnológicos:

- **Virtualización de la plataforma de recursos**. Esencialmente, consiste en la abstracción de un sistema *hardware* completo, que permite que diversas instancias de sistemas operativos se ejecuten sobre ella.
- **Arquitectura orientada a servicios**. Permite crear de sistemas de información altamente escalables que reflejan el negocio de una organización. Además, también brinda una forma bien definida de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.
- ***Framework* orientado a aplicaciones web**. Consiste en un conjunto de conceptos, prácticas y criterios orientados a resolver ciertos problemas de las aplicaciones web, que sirven como referencia para resolver nuevos problemas de índole similar.
- **Despliegue de *software* de código abierto**. Consiste en el desarrollo de aplicaciones de código abierto, que puedan ser utilizadas y ampliadas por una comunidad de usuarios.
- **Servicios web estandarizados**. Consiste en el uso de servicios web que sigan los estándares establecidos. Así, se facilita la incorporación de dichos servicios a las aplicaciones.
- **Uso de sistemas autónomos**. Un sistema autónomo es un conjunto de redes IP que poseen una política de rutas propia e independiente. Cada sistema autónomo gestiona el tráfico que fluye entre él y los restantes sistemas autónomos que forman internet.
- **Computación *grid***. Permite utilizar de forma coordinada recursos de tipos muy diversos (como recursos cómputo, de almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado.

Capítulo 3

Virtualización *hardware*

3.1. Conceptos básicos

La virtualización de *hardware* permite ejecutar múltiples sistemas operativos en una misma máquina. Todos ellos estarán aislados unos de otros, y se ejecutarán sobre una máquina virtual.

Una máquina virtual es la implementación *software* de un computador, y es capaz de ejecutar programas programas como una máquina física. Las máquinas virtuales no accederán directamente al *hardware* de la máquina física en la que residen: para ello, se comunican con el *hipervisor*.

Un *hipervisor* es un programa que crea máquinas virtuales, controla la forma en que estas acceden al *hardware* de la máquina física y gestiona la ejecución de los sistemas operativos de las máquinas virtuales.

3.2. Esquemas de virtualización *hardware*

Existen tres tipos de virtualización *hardware*: virtualización completa, paravirtualización y virtualización asistida por *hardware*.

3.2.1. Virtualización completa

En este caso, la máquina virtual simula todo el repertorio de instrucciones de la CPU, las operaciones de entrada/salida, las interrupciones, los acceso a memoria y, en general, todo el *hardware* que utilice el sistema operativo que se ejecuta sobre ella. Esto evita la necesidad de modificar el sistema operativo de la máquina virtual.

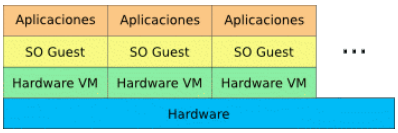

FIGURA 3.1: Virtualización completa

El gran inconveniente de esta alternativa es la necesidad de traducir las instrucciones de la máquina virtual a instrucciones de la máquina física que la alberga (**anfitrión** o ***host***), lo cual requiere mucho tiempo.

3.2.2. Paravirtualización

Si ciertas operaciones se ejecutan directamente sobre la máquina física y no sobre la máquina virtual, su rendimiento puede mejorar de forma considerable. Asimismo, la implementación del sistema de virtualización pasa a ser mucho más sencilla y eficiente. Para hacer esto posible, es necesario

- utilizar una capa adicional, a la que llamaremos hipervisor, para gestionar el acceso a los recursos de la máquina anfitrión, y
- utilizar un sistema operativo modificado en la máquina virtual, que interactuará con el hipervisor para acceder al *hardware*.

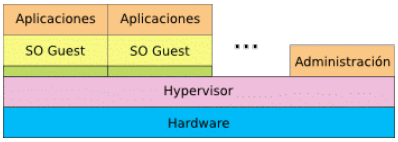

FIGURA 3.2: Paravirtualización

Como podemos observar en la figura 3.2, el sistema operativo de la máquina anfitrión también se ejecutará sobre el hipervisor, y se utilizará para realizar tareas de administración (como, por ejemplo, la creación y destrucción de máquinas virtuales).

Finalmente, si el sistema operativo de la máquina virtual no ha sido modificado, se procederá como en el caso de la virtualización completa y se simulará todo el *hardware* de la misma, con la degradación de rendimiento que esto supone.

3.2.3. Virtualización asistida por *hardware*

Este esquema explota las características de las CPUs actuales para mejorar resolver los problemas de rendimiento de la virtualización completa. Así, permite que un sistema operativo sin modificar pueda ejecutarse sobre la máquina virtual con un rendimiento muy similar al de la paravirtualización.

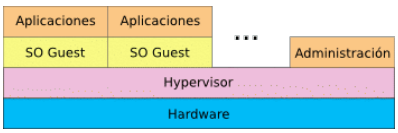

FIGURA 3.3: Virtualización asistida por *hardware*

3.3. Software de virtualización

En la actualidad, existen muchas soluciones de virtualización en el mercado: *VMWare*, *VirtualBox*, Xen, KVM,...

Considerando la actual coyuntura económica, decidimos utilizar una solución de virtualización lo más eficiente posible y gratuita. Estas soluciones permiten implantar *CygnusCloud* con coste cero, ya que no es necesario afrontar el coste de ninguna licencia de *software* y, por el uso que hacen de los recursos, permiten utilizar servidores de hasta seis años de antigüedad para albergar las máquinas virtuales.

Así pues, descartamos dos soluciones de virtualización muy utilizadas (*VMWare* y *VirtualBox*) y nos centramos en la evaluación de las dos soluciones de virtualización gratuitas que ofrecen el mayor rendimiento: Xen y KVM.

3.4. Xen

El hipervisor Xen soporta paravirtualización y virtualización asistida por *hardware* si la CPU de la máquina anfitrión dispone de las instrucciones correspondientes.

Para comprender el funcionamiento de Xen, es fundamental saber cómo gestiona los recursos de las máquinas virtuales. Esto es lo que estudiaremos en la presente sección.

3.4.1. Dominios

En Xen, las máquinas virtuales se llaman **dominios** (en inglés, *domain*). Podemos distinguir dos tipos de dominios:

- el **dominio cero**. Se trata de una máquina virtual privilegiada que dispone de acceso completo al *hardware* y cuyo sistema operativo suministra los *drivers* que se utilizarán para interactuar con este.

  El dominio cero se creará durante el arranque del equipo, y se utilizará para realizar las tareas de administración del servidor.

- el resto de dominios. Estos son las máquinas virtuales que se crean y destruyen durante el funcionamiento del servidor.

Es importante notar que, aunque puede existir un número arbitrario de dominios en la misma máquina, sólo puede haber un único dominio cero.

3.4.2. Tratamiento de interrupciones

Cuando el *hardware* de la máquina anfitrión genera una interrupción, el hipervisor Xen la intercepta y la planifica. Cuando la máquina virtual que la debe atender vuelva a tener acceso a la CPU, realizará el tratamiento correspondiente.

Esto permite que, siempre que se genere una interrupción, no sea necesario esperar a que las máquinas virtuales la procesen, lo cual es muy costoso.

3.4.3. Tiempo de CPU

El tiempo de CPU es el recurso más utilizado. Para repartirlo, Xen trata a los distintos dominios como un sistema operativo trataría a los programas, es decir, utiliza un planificador que garantiza que todos los dominios pueden acceder a la CPU.

3.4.4. Memoria

El hipervisor, como gestor de los recursos de la máquina, tiene acceso a toda la memoria de la máquina anfitrión. En ella, se aloja el contenido de la memoria física de cada máquina virtual y la tabla de páginas de Xen. Es importante notar que cada máquina virtual gestiona su propia memoria virtual.

Como dispone de su propia tabla de páginas, Xen puede controlar las direcciones de memoria físicas a las que acceden las distintas máquinas virtuales. De esta manera, una máquina virtual sólo podrá acceder a la memoria que Xen le ha asignado.

3.4.5. Dispositivos de entrada/salida

Para manipular los dispositivos de entrada/salida, Xen delega en los drivers que forman parte del sistema operativo del dominio cero.

Por otra parte, las máquinas virtuales accederán a los distintos dispositivos de entrada/salida de la máquina anfitrión a través de los dispositivos virtuales, que son enlaces entre las interfaces de las máquinas virtuales y los dispositivos de la máquina física.

Finalmente, para aumentar el rendimiento, el acceso a los dispositivos se hace mediante un *buffer* en anillo, por lo que los dominios pueden utilizarlos directamente sin que sea necesaria la intervención del hipervisor.

3.4.6. Red

Como sucede con el resto de dispositivos, Xen proporciona una interfaz de red virtual a los dominios. Asimismo, Xen implementa las funciones que permiten el flujo de tramas MAC entre las interfaces de red virtuales y las interfaces de red de la máquina anfitrión.

Es importante notar que Xen no valida las tramas MAC que se transfieren entre estas interfaces. Para ello, delega en el sistema operativo del dominio cero.

3.4.7. Dispositivos de bloques

La gestión de los dispositivos de bloques que hace Xen es bastante similar a la gestión de los dispositivos de red.

En este caso, el hipervisor proporciona un dispositivo de bloques virtual a las distintas máquinas virtuales, y delega en el sistema operativo del dominio cero para mapear los comandos del dispositivo de bloques virtual sobre el dispositivo de bloques físico.

3.5. KVM

Como sabemos, el *kernel Linux* es capaz de repartir los recursos de una máquina entre los distintos procesos que se ejecutan sobre la misma. Por ello, incluyendo un reducido conjunto de modificaciones y haciendo que las máquinas virtuales se ejecuten como procesos, es posible lograr que el propio *kernel Linux* se comporte como un hipervisor. Esta es la idea fundamental del hipervisor KVM (*Kernel-based Virtual Machine*), que se distribuye como un módulo del *kernel*.

A diferencia de lo que ocurre en el caso de Xen, la CPU de la máquina anfitrión debe incluir soporte para la virtualización asistida por *hardware*. Por ello, en la actualidad, KVM sólo puede utilizarse con CPUs x86 que cuenten con las extensiones de virtualización *Intel* VT o AMD-V.

3.5.1. Funciones soportadas por KVM

3.5.1.1. Seguridad

KVM puede utilizar los mecanismos de seguridad estándar integrados con el kernel *Linux*: SELinux y AppArmor.

Por ejemplo, el proyecto SVirt, construido sobre SELinux, refleja el esfuerzo de la comunidad para integrar el control de acceso obligatorio (en inglés, *Mandatory Access Control*) con KVM.

3.5.1.2. Gestión de memoria

La memoria utilizada por una máquina virtual se gestionará de la misma forma que la memoria utilizada por el resto de procesos. Así, podrá ser guardada en disco (*swapped*) y utilizada a través de páginas grandes (*large pages*).

Además, KVM soporta las últimas características de virtualización en memoria proporcionadas por fabricantes, como EPT (*Extended Page Table*) de Intel o RVI (*Rapid Virtualization Indexing*) de AMD.

3.5.1.3. Migraciones en caliente

KVM soporta migraciones en caliente (en inglés, *live migrations*). Las migraciones en caliente consisten en mover una máquina virtual en ejecución a otro servidor, desde el que seguirá ejecutándose.

Estas migraciones son totalmente trasparentes para el usuario: mientras la máquina se realoja en otro servidor físico permanece encendida, con sus conexiones de red activas y sus aplicaciones en ejecución.

3.5.1.4. Gestión del almacenamiento

Para almacenar las imágenes de disco de las máquinas virtuales, es posible utilizar cualquier dispositivo de almacenamiento soportado por el kernel *Linux*. Esto incluye discos duros locales (IDE/SCSI/SATA), servidores NAS (*Network Attached Storage*) o redes SAN (*Storage Area Network*). Asimismo, también es posible utilizar sistemas de ficheros distribuidos como GFS2, OCFS o ClusterFS.

De forma nativa, KVM soporta los formatos de imagen RAW, `qcow`, `qcow2` y `vmdk` (el formato nativo de *VMWare*). Los tres últimos soportan compresión, por lo que el espacio libre en las imágenes de disco que los utilizan no ocupará espacio en disco en la máquina anfitrión.

Finalmente, existen dos alternativas para ubicar las imágenes de disco:

- **almacenamiento local**. En este caso, las imágenes de disco se encuentran en la propia máquina anfitrión. Esto resulta muy útil para entornos de desarrollo, pruebas y pequeños despliegues, aunque no se recomienda en entornos con muchas máquinas virtuales o en los que se realicen migraciones en caliente.
- **almacenamiento en red**. En este caso, las imágenes de disco no se encuentran en la máquina anfitrión, y se accede a ellas a través de una red.

3.5.1.5. Red en KVM

En KVM, las interfaces de red de las máquinas virtuales pueden operar en tres modos principales:

- modo ***bridge***. En este caso, la máquina virtual se conecta directamente a la red que la que forma parte la máquina anfitrión. Desde el exterior, parecerá que la máquina virtual es un equipo físico conectado a esa red.
- modo **NAT** (*Network Address Translation*, traducción de direcciones de red). En este caso, se crea una red virtual dentro de la máquina anfitrión, a la que se conectarán todas las máquinas virtuales. Desde el exterior, esta red no será accesible, y parecerá que la máquina anfitrión es la que genera el tráfico de las máquinas virtuales.
- modo ***route***. En este caso, las máquinas virtuales se encuentran en una subred virtual, a la que es posible acceder desde el exterior (a través de su encaminador predeterminado).

3.5.2. *Drivers*

El hipervisor KVM utiliza virtualización híbrida, combinando la paravirtualización y la virtualización asistida por *hardware*.

Por ello, en los sistemas operativos de las máquinas virtuales hay que instalar drivers de paravirtualización. Estos permiten a dicho sistema operativo utilizar la interfaz de entrada/salida para dispositivos de bloques y dispositivos de red.

En la actualidad, los drivers de paravirtualización más utilizados son los del proyecto VirtIO, desarrollado por IBM y *RedHat*. El objetivo principal de este proyecto es mejorar el rendimiento de las máquinas virtuales a través una mejor interacción entre KVM y los sistemas operativos de las mismas.

3.5.3. Ventajas e inconvenientes de KVM

El uso de KVM tiene las siguientes ventajas:

- su instalación es muy sencilla: basta con incorporar un nuevo módulo al kernel *Linux*.
- permite la migración en caliente de las máquinas virtuales.
- los servidores que albergan las máquinas virtuales se administran como máquinas Linux convencionales.
- las máquinas virtuales se tratan como procesos, por lo que resulta muy sencillo manipularlas.

Por otra parte, en la actualidad el uso de KVM también tiene asociados varios inconvenientes:

- se trata de un proyecto con poco recorrido comercial. Su reciente aparición en el mercado hace que no disponga el mismo soporte que otras soluciones, como Xen y *VMWare*.

- no existen herramientas sofisticadas para gestionar servidores y máquinas virtuales.
- deben introducirse mejoras en áreas como el soporte del almacenamiento virtual, la seguridad, la alta disponibilidad, la tolerancia a fallos y la gestión del consumo energético.

3.5.4. Limitaciones de KVM

El uso del hipervisor KVM tiene varias limitaciones, que afectan tanto al sistema operativo de la máquina virtual como al servidor que aloja las máquinas virtuales.

Estas limitaciones están asociadas a

- los *overcommits*. KVM permite que las máquinas virtuales utilicen más memoria, más CPUs o más espacio en disco del disponible en el servidor. Si bien esto permite atender a más usuarios, también puede dar lugar a un empeoramiento del rendimiento y a fallos en las máquinas virtuales.
- la generación de las direcciones MAC. Si una interfaz de red virtual no tiene asignada una dirección MAC, KVM generará dicha dirección MAC. Esto puede dar lugar a problemas en la red cuando más de una interfaz de red virtual recibe la misma dirección MAC.
- la suspensión y la hibernación del servidor de maquinas virtuales. Mientras existan máquinas virtuales activas, no es posible suspender o hibernar un servidor de máquinas virtuales.

3.6. Xen frente a KVM

En este apartado, justificaremos por qué decidimos utilizar el hipervisor KVM en lugar del hipervisor Xen.

3.6.1. Integración en el *kernel*

KVM es el único hipervisor que, a día de hoy, se encuentra completamente integrado en el *kernel Linux*. KVM utiliza el *kernel Linux* para todo, desde la gestión de dispositivos de entrada/salida hasta el reparto del tiempo de CPU y la gestión de la memoria.

Sin embargo, Xen todavía no ha sido aceptado como parte del núcleo *Linux*, y por ello no puede aprovechar algunas funciones muy útiles que este ofrece, tales como los algoritmos de planificación CFS (*Completely Fair Scheduler*), la gestión de la tabla de páginas o la sobreexplotación de la memoria con KSM (*Kernel Samepage Merging*).

Además, a diferencia de lo que ocurre en el caso de KVM, Xen requiere modificar sustancialmente el proceso de arranque de las máquinas en las que está instalado: en él, es necesario cargar el hipervisor y crear el dominio cero.

En definitiva, KVM resulta mucho más fácil de instalar y configurar que Xen: basta con instalar un módulo del *kernel* más durante el proceso de arranque.

3.6.2. Rendimiento

El rendimiento de Xen y KVM es, en términos generales, bastante similar. No obstante,

- cuando en las máquinas virtuales utilizan la CPU de forma intensiva, el rendimiento de Xen es superior al de KVM, y
- cuando las máquinas virtuales realizan muchas operaciones de entrada/salida a disco, el rendimiento de KVM es superior al de Xen.

Para llegar a esta conclusión, en el estudio [12] se compiló el *kernel Linux* y se ejecutó el *benchmark IOZone* directamente sobre distintas máquinas físicas y sobre máquinas virtuales con diversas características. La primera prueba se caracteriza por el uso intensivo de CPU, y la segunda se caracteriza por realizar un gran número de operaciones de entrada/salida a disco.

Los resultados obtenidos fueron los siguientes:

- en la primera prueba, la pérdida de rendimiento con respecto a la máquina original fue, en promedio, de un 1 % en las máquinas virtuales Xen, y de un 36 % en las máquinas virtuales KVM.
- en la segunda prueba, la pérdida de rendimiento con respecto a la máquina original fue, en promedio, de un 42,9 % en las máquinas virtuales Xen, y de un 3,6 % en las máquinas virtuales KVM.

3.6.3. Soporte técnico

Gracias a su integración en el *kernel*, las versiones de KVM se publican con mucha frecuencia, lo que garantiza la rápida corrección de los *bugs* que se detecten.

Asimismo, *RedHat* e IBM están realizando una importante apuesta por KVM, publicando y actualizando numerosas páginas de documentación y guías de migración a KVM desde otras soluciones de virtualización como VMWare y Xen.

Por otra parte, la actualización del código de la versión *open-source* de Xen sólo tiene lugar una vez cada año, por lo que el soporte de Xen es bastante peor que el de KVM.

Además, la documentación de la versión *open-source* de Xen es muy escasa y frecuentemente no está actualizada a la última versión.

En definitiva, puesto que

- la instalación y uso de KVM son mucho más sencillos que los de Xen,
- el rendimiento de ambas soluciones es prácticamente idéntico, y
- KVM cuenta con mucho mejor soporte que Xen

decidimos utilizar el hipervisor KVM en lugar del hipervisor Xen.

Capítulo 4

Arquitectura del sistema

4.1. Introducción

En este capítulo presentaremos una visión general de la arquitectura de *CygnusCloud*. Para facilitar su comprensión, hemos intentado que la claridad prevalezca a la hora de exponer los distintos conceptos y decisiones de diseño.

Por ello, evitaremos a toda costa realizar descripciones exhaustivas de todas y cada una de las clases y de sus distintos atributos. Si el lector desea conocer con total precisión qué atributos y qué métodos tiene cada clase, cómo se comporta cierto método o cuál es la finalidad de un determinado atributo, le remitimos al código fuente, extensamente comentado.

Por otra parte, en este capítulo también prestaremos especial atención a las decisiones de diseño que hemos tomado ya que, a nuestro parecer, son fundamentales para comprender el funcionamiento del sistema, sus ventajas y sus limitaciones.

4.1.1. Visión general

Los contenidos de este capítulo se agrupan en las siguientes secciones:

1. La presente **introducción**.
2. **Objetivos de la arquitectura y restricciones**. En esta sección expondremos todos los objetivos que se tuvieron al diseñar *CygnusCloud*, y también recopilaremos las restricciones con las que hemos tenido que tratar a lo largo de todo el proceso de diseño.
3. **Decisiones de diseño**. En esta sección justificaremos las decisiones más relevantes que hemos tomado en el proceso de diseño.
4. **Vista lógica**. En esta sección, mostraremos cómo las funciones de cada subsistema se distribuyen entre sus distintos módulos con la ayuda de un diagrama de paquetes.

   Además, utilizaremos diagramas de clase y diagramas de secuencia para exponer las responsabilidades de cada elemento significativo de la arquitectura y sus relaciones e interacciones con el resto de componentes de la misma.
5. **Vista de despliegue**. En esta sección mostraremos cómo se distribuyen los distintos subsistemas que forman parte de *CygnusCloud* entre las distintas máquinas que forman parte de la infraestructura utilizada.
6. **Vista de implementación**. Concluiremos el presente capítulo con esta sección, en la que describiremos los componentes que se distribuyen con cada subsistema.

4.1.2. Sobre los diagramas UML de este documento

Para que el aumento del tamaño de los diagramas UML en un lector de ficheros PDF no suponga una pérdida de calidad de los mismos, hemos procurado incrustarlos en este documento como ficheros `.pdf`.

El comando de *IBM Rational Software Architect 7.5* que los genera usa degradados para mejorar la presentación de los diagramas, haciendo que los elementos que aparecen en la parte superior del diagrama sean más claros que los que aparecen en la parte inferior del mismo.

En cualquier caso, este efecto *no* tiene asociado ningún tipo de significado en los diagramas.

4.2. Objetivos de la arquitectura y restricciones

En esta sección describiremos todos los objetivos que consideramos a la hora de diseñar la arquitectura, así como las restricciones que hemos tenido en cuenta a lo largo de todo el proceso de diseño.

4.2.1. Objetivos

Con *CygnusCloud* queremos hacer posible que cualquier estudiante de la Universidad Complutense, sea de la titulación que sea, pueda sacar el máximo partido a cualquier aula de informática infrautilizada del campus. Para ello, es necesario que

- el sistema pueda atender a más alumnos utilizando más servidores y más ancho de banda. Por tanto, la *escalabilidad* debe considerarse en el diseño.
- los alumnos y profesores puedan utilizar *CygnusCloud* sin necesidad de instalar ningún *software* adicional en las aulas de informática. De no ser así, no es posible aprovechar los equipos de las aulas que no tienen instalado el *software* adicional.

Por otra parte, dada la actual situación de crisis económica, *CygnusCloud* debe poder implantarse con *coste cero*. Para ello,

- debemos utilizar exclusivamente software gratuito, y
- el *software* debe ejecutarse sobre servidores antiguos. Por ello, resulta imprescindible que este sea tan eficiente como sea posible.

Asimismo, el sistema sólo podrá utilizarse con garantías cuando detecte y trate un número suficiente de errores. Por ello, el sistema debe ser *robusto*, de forma que pueda detectar y trate el mayor número de errores posible.

Finalmente, dado que el desarrollo de *CygnusCloud* sólo se extiende a lo largo de un curso académico, también queremos que cualquiera que lo desee pueda continuarlo. Por ello, hemos procurado que nuestro diseño también se caracterice por su *simplicidad*.

4.2.2. Restricciones

Tal y como mencionamos en el apartado 4.2.1, *CygnusCloud* debe poder implantarse con coste cero, por lo que debemos construirlo utilizando *software* gratuito. Puesto que los hipervisores deben instalarse en un sistema operativo en concreto, que tiene que ser gratuito, esto ha impuesto la primera restricción importante: la *dependencia de la plataforma Linux*.

Por otra parte, existe una ingente cantidad de distribuciones *Linux*, que utilizan distintos sistemas de gestión de paquetes y distintos procedimientos de configuración.

Para garantizar que *CygnusCloud* funciona correctamente en el mayor número posible de distribuciones *Linux*, resulta fundamental utilizar mecanismos estándar para interactuar con los hipervisores y, a ser posible, lenguajes interpretados. Esto ha impuesto una doble restricción:

- el uso obligatorio de la librería *libvirt* para interactuar con los hipervisores y configurar las redes virtuales, y
- el uso de un lenguaje de tipo interpretado, como *Java* o *Python*, como principal lenguaje de programación.

Además, la práctica totalidad de los *frameworks* que hemos evaluado al diseñar la web se basan en el modelo vista-controlador o en otros patrones de diseño derivados de él, lo que nos ha obligado a utilizar dicho patrón de diseño.

Asimismo, puesto que la red troncal de la Complutense utiliza la versión 4 del protocolo IP (*Internet Protocol*), *CygnusCloud* debe operar en redes IP versión 4.

No obstante, dado que esta versión del protocolo IP ha quedado totalmente obsoleta, hemos decidido facilitar, en la medida de lo posible, la transición a la versión 6 del protocolo IP, de forma que sólo sea necesario introducir pequeños cambios en el código del módulo de comunicaciones y ajustar las longitudes de ciertos campos en los esquemas de las bases de datos.

Finalmente, para reducir el tiempo de implementación de *CygnusCloud* resulta imprescindible que *todos los subsistemas compartan tanta funcionalidad como sea posible*.

De esta manera, todos ellos utilizarán el mismo código para realizar, entre otras cosas, operaciones básicas sobre una base de datos y comunicaciones a través de una red.

No obstante, esto sólo es posible si *todos los subsistemas se implementan utilizando el mismo lenguaje de programación*, lo que ha restringido considerablemente el número de librerías y *frameworks* que hemos podido considerar.

4.3. Terminología básica

En este capítulo, utilizamos frecuentemente los siguientes términos:

- **familia de máquinas virtuales**: conjunto de máquinas virtuales que tienen las mismas características y que utilizan la misma familia de sistemas operativos.

  Por ejemplo, todas las máquinas virtuales que utilicen el sistema operativo *Linux*, tengan una CPU, 512 *megabytes* de memoria RAM y 10 GB de espacio en disco pertenecerán a la misma familia. Análogamente, todas las máquinas que utilicen el sistema operativo *Windows*, tengan tres CPUs, 3 *gigabytes* de memoria RAM y 30 GB de espacio en disco pertenecerán a otra familia distinta.

- **configuración** o **imagen**: contenido de *todos* los discos duros asociados a una máquina virtual. La imagen de una máquina virtual determinará el operativo que utiliza y la forma en que los usuarios podrán trabajarán con ella.

  Por ejemplo, si al crear una máquina virtual le "conectamos" un disco duro virtual con el sistema operativo *Windows* con un entorno de desarrollo integrado instalado, la máquina utilizará dicho sistema operativo, y en ella podrá utilizarse dicho entorno de desarrollo. Si, en cambio, le "conectamos" un disco duro virtual con el sistema operativo *Linux* y una *suite* ofimática instalada, la máquina utilizará el sistema operativo *Linux*, y en ella podrá utilizarse la *suite* ofimática.

- **instancia de una imagen**: máquina virtual que utiliza una imagen.

- **configuración base**, **configuración *vanilla***, **imagen base** o **imagen *vanilla***: configuración que sólo contiene un sistema operativo y las herramientas más básicas instaladas.

- **anfitrión** o ***host* de una máquina virtual**: servidor en el que reside cierta máquina virtual.

- **anfitrión** o ***host* de una imagen**: servidor que puede albergar instancias de una imagen.

- **despliegue**: proceso durante el cual un servidor pasa a almacenar una configuración o imagen. Cuando finaliza, ese servidor podrá albergar máquinas virtuales que utilicen la imagen que se despliega.

- ***cluster***: agrupación de servidores que se utilizan y administran de forma conjunta.

- **infraestructura**: conjunto formado por todos los *clusters* que utiliza el sistema *CygnusCloud*.

- **imagen de disco**: fichero almacenado en el disco duro de la máquina anfitrión, en el que se almacenan los datos de un disco duro asociado a cierta máquina virtual.

- **disco duro virtual**: unidad de disco duro de una máquina virtual. Sus datos se almacenarán en una imagen de disco.

4.4. Decisiones de diseño

4.4.1. Uso de una aplicación web para interactuar con *CygnusCloud*

En la sección 4.2.1 dijimos, entre otras cosas, que *CygnusCloud* debe poder utilizarse sin necesidad de instalar *software* adicional en las aulas de informática. Para ello, existen dos alternativas: crear una aplicación web o crear una aplicación de escritorio convencional que pueda utilizarse sin necesidad de ser instalada.

El uso de una aplicación de escritorio tiene una gran ventaja: no es necesario utilizar tecnologías web, totalmente desconocidas para nosotros al inicio del proyecto. No obstante, también tiene asociados algunos inconvenientes considerables:

- para utilizar *CygnusCloud* en móviles o tabletas es necesario crear una una aplicación específica, muy dependiente de la plataforma del dispositivo.
- puesto que los usuarios no tienen por qué descargarse siempre la aplicación antes de utilizar *CygnusCloud*, es posible que convivan varias versiones de esta, lo que puede dar lugar a conflictos y, sobre todo, a problemas de seguridad.

  La única forma de resolver estos problemas es creando un sistema de actualizaciones periódicas, por lo que el diseño y la implementación de la aplicación se complican.
- para poder enviar las peticiones al sistema, la aplicación necesita averiguar la dirección IP y el puerto de la máquina que las tratará. Para ello, existen dos alternativas:
  - incrustar estos datos en el código. Esto dificulta el cambio de la IP y el puerto de la máquina que tratará las peticiones.
  - anunciarlos periódicamente a través de la red. Con esto, además de desperdiciar ancho de banda, estamos comprometiendo la seguridad del sistema, ya que cualquier usuario malintencionado podrá recibir estos mensajes.
- aunque el tráfico viaje cifrado, cualquier usuario que utilice un analizador de tráfico podrá averiguar la dirección IP y el puerto de la máquina que trata sus peticiones, lo que facilita mucho la realización de ataques de denegación de servicio.

Por otra parte, al utilizar una aplicación web desaparecen muchos de estos inconvenientes, ya que

- para utilizar *CygnusCloud* en móviles o tabletas sólo es necesario introducir cambios en el *layout* o disposición de elementos de la página.
- no es necesario descargar ningún programa en los PCs que usan los usuarios. Esto hace el sistema más fácil de usar (basta con utilizar un navegador web) y permite ahorrar ancho de banda.
- las actualizaciones son muy fáciles de aplicar: basta con parar el servidor web, copiar en él los ficheros binarios y reiniciarlo.
- aunque sigue siendo posible realizar ataques de denegación de servicio, estos sólo afectarán al servidor web y no a los servidores de la infraestructura, por lo que esta seguirá procesando las peticiones ya recibidas correctamente.

Finalmente, el uso de una aplicación web tiene un serio inconveniente: debido a nuestra inexperiencia en este campo, es posible que no detectemos algunas alternativas para comprometer la seguridad del sistema.

De todas formas, podemos considerar que la red troncal de la UCM es segura, por lo que este tipo de prácticas no deberán preocuparnos excesivamente.

4.4.2. Descomposición del sistema *CygnusCloud* en cuatro subsistemas

Para atender las peticiones de los usuarios, en la infraestructura de *CygnusCloud* colaboran hasta cuatro subsistemas distintos.

En esta sección, mostraremos, de forma general, las responsabilidades de cada uno de ellos. Además, también expondremos los motivos que nos llevaron a hacer esta descomposición.

4.4.2.1. Los servidores de máquinas virtuales

Instanciar una máquina virtual requiere muchos recursos. No basta con asignarle memoria RAM y tiempo de CPU: también es necesario reservar espacio en disco en el que almacenar los datos de los usuarios y los archivos de paginación.

Por muy potente que sea el *hardware* del servidor que alojará las máquinas virtuales, el número máximo de máquinas virtuales estará siempre limitado. Por tanto, la única forma de superar esta limitación y aumentar el número de máquinas virtuales que pueden estar activas a la vez es utilizando varios servidores para albergarlas.

Por otra parte, el arranque de máquinas virtuales y la creación o edición de imágenes son operaciones que requieren mucha entrada/salida. Para garantizar que el tiempo de respuesta del sistema es adecuado, lo más conveniente es que los nodos que albergan máquinas virtuales se dediquen a ello en exclusiva. De ahora en adelante, llamaremos **servidores de máquinas virtuales** a los nodos que albergan exclusivamente máquinas virtuales.

Es importante notar que, una vez creadas las las máquinas virtuales, el servidor VNC que utilizarán los usuarios para interactuar con ellas reside en el propio servidor de máquinas virtuales, por lo que el tráfico VNC circulará exclusivamente entre los PCs de los usuarios y los servidores de máquinas virtuales.

Además, el tráfico que los usuarios generan para conectarse a internet se vuelca directamente a la red, es decir, no atraviesa ninguna otra máquina de la infraestructura de *CygnusCloud*.

4.4.2.2. El repositorio de imágenes

Para arrancar máquinas virtuales y crear o editar configuraciones, las imágenes de disco correspondientes deben estar desplegadas en un servidor de máquinas virtuales. No obstante, hay imágenes de disco que no deben estar desplegadas en ningún servidor de máquinas virtuales. Este es el caso de

- las imágenes base o imágenes *vanilla*, y
- las imágenes de disco que no están en uso y no han terminado de modificarse.

En ambos casos, el *software* de la imagen no está completamente configurado, por lo que ningún usuario podrá utilizarlo para trabajar. Así, la presencia de estas configuraciones en un servidor de máquinas virtuales supone un desperdicio de espacio en disco.

Además, para simplificar la administración del sistema no resulta conveniente que las imágenes de disco estén diseminadas entre múltiples máquinas. Si una única máquina del *cluster* almacena todas las imágenes existentes, para realizar modificaciones en cualquiera de ellas bastará con acudir a un único lugar y, posteriormente, actualizar la configuración en el resto de máquinas del *cluster*.

Por tanto esta máquina, a la que llamaremos **repositorio de imágenes**, almacenará una copia de todas las imágenes de disco existentes. Asimismo, el repositorio de imágenes será la única que cuente con una copia las imágenes *vanilla* y de las imágenes de disco que no están en uso y no han terminado de modificarse.

4.4.2.3. El servidor de *cluster*

En un momento dado, varios servidores de máquinas virtuales pueden albergar una nueva instancia de cierta máquina virtual. Ahora bien, no todos ellos tienen por qué encontrarse en las mismas condiciones.

Por ejemplo, si los servidores A y B pueden albergar una nueva instancia de la máquina virtual Debian-LSO y el servidor A alberga más instancias o máquinas virtuales que el servidor B, la experiencia del usuario mejorará si alojamos la nueva instancia en el servidor B.

Por tanto, necesitamos utilizar una máquina que

- recopile periódicamente el estado de todas las máquinas del *cluster*, y
- utilice esta información para realizar el balanceado de carga entre dichas máquinas.

Por otra parte, para ahorrar ancho de banda resulta fundamental que, al atender una petición, se compruebe si el *cluster* la puede satisfacer.

Por ejemplo, si el repositorio de imágenes no tiene suficiente espacio en disco para alojar las modificaciones de una configuración, no tiene sentido transferir estos ficheros a un servidor de máquinas virtuales y alojar en él la máquina virtual que se usará para hacer las modificaciones, ya que los ficheros correspondientes no podrán subirse al repositorio de imágenes.

Además, para facilitar la administración del sistema, esta máquina también se ocupará de arrancar, parar, dar de alta y borrar servidores de máquinas virtuales. De ahora en adelante, la llamaremos **servidor de *cluster***.

Como el lector habrá notado ya, el servidor de *cluster* se convierte un cuello de botella que limita el número máximo de servidores de máquinas virtuales del *cluster* y, por ende, el número máximo de usuarios que es posible atender.

Para superar esta limitación y atender a un número mayor de usuarios, podemos utilizar varios *clusters,* cada uno de los cuales tendrá asociado su propio servidor de *cluster*, uno o más servidores de máquinas virtuales y su propio repositorio de imágenes.

Es importante notar que, ya que cada *cluster* tiene su propio repositorio de imágenes, es completamente autónomo, por lo que no tiene por qué compartir configuraciones ni familias de máquinas virtuales con otros *clusters*.

4.4.2.4. El servidor web

Como los servidores de *cluster* son ya un cuello de botella y puede haber varios, la página web de *CygnusCloud* no puede residir en ninguno de ellos.

Además, por motivos de eficiencia, los servidores de *cluster* no realizan ningún tipo de control de acceso, por lo que no pueden atender directamente las peticiones de los usuarios.

Así pues, es necesario utilizar una máquina adicional, el **servidor web**, cuyas principales funciones son las siguientes:

- alojar la aplicación web de *CygnusCloud*, que se usará para interactuar con la infraestructura.
- realizar el control de acceso.
- redirigir las peticiones de cada usuario al servidor de *cluster* que debe atenderlas.

4.4.3. Implementación de una infraestructura *ad-hoc*

Para gestionar los servidores de la infraestructura y las máquinas virtuales, es posible utilizar dos soluciones gratuitas y muy populares: *OpenStack* y *OpenNebula*.

Estas herramientas permiten gestionar *clouds* de gran tamaño, e implementan toda la funcionalidad que necesitamos para realizar la gestión de los servidores de la infraestructura y de las máquinas virtuales.

No obstante, hemos preferido diseñar e implementar una infraestructura *ad-hoc* por las siguientes razones:

- *OpenStack* y *OpenNebula* proporcionan muchas funciones a costa de consumir muchos recursos. Tal y como dijimos en la sección 4.2.1, *CygnusCloud* debe poder implantarse con coste cero, y para ello es imprescindible aprovechar servidores antiguos, para los que estas soluciones pueden ser demasiado pesadas.

- *OpenStack* y *OpenNebula* son excesivamente grandes. Para implementar *CygnusCloud*, sólo es necesario utilizar un pequeño subconjunto de la funcionalidad que ofrecen *OpenStack* y *OpenNebula*.

  Ahora bien, para fijar dicho subconjunto y familiarizarnos con estas soluciones habríamos necesitado, como mínimo, tres meses. Esto supone casi la tercera parte del tiempo de desarrollo del proyecto, por lo que no es aceptable.

- al implantar una infraestructura *ad-hoc*, podemos descubrir por nosotros mismos qué hace falta para suministrar un servicio del tipo infraestructura como servicio. Así, nos podemos familiarizar con muchas de las tecnologías de virtualización existentes en el mercado.

  Además, este conocimiento nos permitirá, en un futuro próximo, comprender mejor y más rápidamente el funcionamiento de *OpenStack* y *OpenNebula*.

4.4.4. Configuración de las redes virtuales en modo NAT

Para que los usuarios de *CygnusCloud* puedan trabajar en las máquinas virtuales, resulta imprescindible que estas accedan, a través de la red troncal de la Complutense, al Campus Virtual y a internet.

La forma más sencilla de lograr esto es configurando un *bridge* para cada máquina virtual. Sin entrar demasiado en detalles, podemos decir que esta solución permite que las máquinas virtuales se conecten a la red troncal de la Complutense como si fuesen PCs convencionales.

En la sección 4.2.2 dijimos que la red troncal de la UCM sigue utilizando la versión 4 del protocolo IP (*Internet Protocol*).

Con esta primera alternativa, cada máquina virtual debe tener una dirección IP única. Como la red troncal de la Complutense es totalmente plana y las direcciones IP versión 4 son muy escasas, el número de máquinas virtuales que pueden estar activas a la vez estará limitado, ya que

- dado que *CygnusCloud* debe poder implantarse con coste cero, no es posible adquirir nuevos (y costosos) bloques de direcciones IP para utilizarlo.
- las máquinas virtuales de *CygnusCloud* no pueden monopolizar el uso de las direcciones IP disponibles: esto comprometería el normal funcionamiento de la red troncal de la UCM.

Así pues, esta primera alternativa no es viable. Para superar sus limitaciones, hemos decidido utilizar NAT (*Network Address Translation*, traducción de direcciones de red) en los servidores de máquinas virtuales. En este caso,

- sólo los servidores de máquinas virtuales (y no las máquinas virtuales) se conectan a la red troncal de la UCM.
- las máquinas virtuales se conectan a una red virtual, existente sólo en el servidor de máquinas virtuales.
- cuando una máquina virtual envíe un datagrama IP, el servidor reescribirá su cabecera. En la red troncal de la UCM, parecerá que es el servidor de máquinas virtuales el que ha creado dicho datagrama.
- cuando el servidor de máquinas virtuales reciba un datagrama IP, reescribirá su cabecera y lo reenviará a la máquina virtual correspondiente. En la máquina virtual, parecerá que se ha accedido directamente a la red troncal de la UCM.

Como cada servidor de máquinas virtuales puede contener varias máquinas virtuales activas a la vez, esta alternativa no tiene los inconvenientes de la anterior.

No obstante, nos hemos asegurado de que resulte sencillo crear un *bridge* para cada máquina virtual. Así, los administradores del sistema podrán utilizar una u otra alternativa cuando lo estimen oportuno.

4.4.5. Uso del protocolo de escritorio remoto VNC

Para que los usuarios puedan interactuar con la máquina virtual, es imprescindible utilizar un protocolo de escritorio remoto. Estos permiten que los clientes interactúen con las máquinas virtuales a través de las interfaces gráficas de usuario a las que están acostumbrados. En su momento, consideramos dos alternativas: VNC y RDP.

VNC (*Virtual Network Computing*, computación virtual en red) es un protocolo de escritorio remoto independiente de la plataforma y desarrollado por *Olivetti*, *Oracle Corporation* y, posteriormente, por *AT&T*.

Por su parte, **RDP** (*Remote Desktop Protocol*, protocolo de escritorio remoto) es un protocolo de escritorio remoto desarrollado por *Microsoft*, utilizado mayoritariamente en sistemas *Windows*. Existen clientes y servidores RDP para otros sistemas operativos, como *Linux* y *Mac OS X*.

En términos generales, el funcionamiento de ambos protocolos es el siguiente:

- la máquina que comparte su pantalla se denomina **servidor**.
- la máquina en la que trabajan los usuarios se denomina **cliente**.
- cuando el cliente mueve el ratón o pulsa una tecla,

    1. el cliente de escritorio remoto envía los datos correspondientes al servidor de escritorio remoto.
    2. al recibirlos, el servidor de escritorio remoto generará el evento de teclado o ratón correspondiente, que será atendido por el sistema operativo de la máquina remota de la forma habitual.
    3. el servidor de escritorio remoto averigua qué zonas de la pantalla se han actualizado, comprime esta información y se la envía al cliente de escritorio remoto.
    4. a partir de esta información, el cliente de escritorio remoto actualiza la imagen de la pantalla que ve el usuario.

Como *CygnusCloud* debe poder implantarse con coste cero, los clientes y servidores de escritorio remoto que utilicemos deben ser gratuitos.

En términos generales, las implementaciones libres y gratuitas de RDP y VNC tienen un rendimiento bastante similar. Además, VNC está integrado en la práctica totalidad de sistemas de virtualización, por lo que es posible utilizarlo sin necesidad de instalar nada en las máquinas virtuales. Por ello, decidimos utilizar el protocolo VNC.

4.4.6. Uso del gestor de bases de datos MariaDB

Al inicio del proyecto, estábamos mucho más familiarizados con los gestores de bases de datos relacionales que con los no relacionales. Por ello, decidimos utilizar un gestor de bases de datos relacional para manipular todas las bases de datos de la infraestructura.

En la actualidad, los gestores gratuitos de bases de datos relacionales más completos son dos: MySQL y MariaDB.

MySQL es el gestor de bases de datos relacionales de código abierto más extendido, y se utiliza en servicios web con un gran número de usuarios (como *Facebook*, *Wordpress* y *YouTube*) y también en muchos otros sistemas de menor tamaño.

Por otra parte, **MariaDB** es un gestor de bases de datos relacionales cuya popularidad entre las comunidades de *software libre* no ha parado de crecer. En los últimos meses, la mayoría de distribuciones *Linux* han decidido reemplazar MySQL por MariaDB, y organizaciones de gran tamaño como la *Wikimedia Foundation* han decidido migrar sus bases de datos a MariaDB.

Aunque inicialmente MariaDB estaba basado en la versión 5.5 de MySQL, sus nuevas versiones incluyen un gran número de mejoras enfocadas, sobre todo, a conseguir un rendimiento superior al de MySQL.

Además, como MariaDB es totalmente compatible a nivel sintáctico y a nivel binario la versión 5.5 de MySQL, resulta muy sencillo reemplazar MySQL por MariaDB.

Tras examinar lo que ha sucedido tras la compra de *Sun Microsystems* por parte de *Oracle Corporation* en 2009, nos parece que el desarrollo de MySQL ha sufrido un abandono progresivo, de forma que en la actualidad todo parece indicar que, a medio plazo, *Oracle* va a dejar de dar soporte al proyecto.

Considerando nuestras necesidades, MariaDB y MySQL son totalmente equivalentes. Además, el soporte a medio y a largo plazo de MariaDB parece estar garantizado, cosa que no ocurre en el caso de MySQL. Por ello, decidimos utilizar MariaDB.

4.4.7. Uso del hipervisor KVM

Las dos soluciones de virtualización libres y gratuitas que cuentan con el mejor rendimiento son dos: Xen y KVM. En la sección 3.6 hicimos una comparativa de ambas y explicamos que, por contar con un rendimiento muy similar, mejor documentación y mejor soporte que Xen, optamos por utilizar KVM.

4.4.8. Uso del servidor VNC integrado en KVM

Tal y como dijimos en la sección 4.4.5, para poder utilizar el protocolo de escritorio remoto VNC es necesaria la existencia de un servidor VNC, que envíe periódicamente al cliente VNC las actualizaciones de la pantalla de la máquina virtual.

El uso del servidor VNC integrado en KVM tiene dos grandes ventajas:

- no es necesario instalar ni ejecutar código adicional en la máquina virtual, lo que simplifica la creación y configuración de máquinas virtuales.
- el usuario de la máquina virtual no puede detener por error el servidor VNC. Si el servidor VNC se ejecuta en la máquina virtual y el cliente mata su proceso por error, la máquina virtual quedará inutilizada.
- la gestión de los datos de conexión a las máquinas virtuales es muy sencilla.

Por otra parte, el uso de un servidor VNC que resida en cada máquina virtual tiene una gran ventaja: aporta mucha más flexibilidad. No obstante, en este caso la gestión de los datos de conexión es más compleja, ya que estos se fijan en la máquina virtual y son, *a priori*, desconocidos para los servidores de la infraestructura.

Tras hacer pruebas para evaluar las dos alternativas, concluimos lo siguiente:

- la gran mayoría de servidores VNC libres y gratuitos proporcionan las mismas funciones que el servidor VNC integrado en KVM.
- el rendimiento de ambas es prácticamente idéntico.

Además, como el uso del servidor VNC integrado en KVM simplificaba considerablemente la implementación del sistema, finalmente decidimos utilizarlo.

4.4.9. Uso del cliente VNC noVNC

Tal y como dijimos en la sección 4.4.1, *CygnusCloud* se utilizará a través de una aplicación web.

Para que los usuarios puedan utilizar las máquinas virtuales, es necesario disponer de un cliente VNC, que se conecte al servidor VNC correspondiente. Además, este debe integrarse con la aplicación web de forma que, al arrancar una máquina virtual, los usuarios puedan utilizarlo para interactuar con ella.

Inicialmente, consideramos modificar un cliente de escritorio VNC libre y gratuito. Este debía ser multiplataforma y funcionar, como mínimo, en los sistemas operativos *Windows* y *Linux*. En su momento, consideramos tres distintos: TigerVNC, TightVNC y la versión *open source* de RealVNC.

La principal ventaja de esta alternativa es el rendimiento: dado que estos clientes están escritos en C o C++, es posible utilizarlos en equipos poco potentes de forma muy satisfactoria.

No obstante, el hecho de que estos clientes estén escritos en C o C++ supone también su mayor inconveniente: es necesario crear una versión distinta para cada sistema operativo, algo que no es trivial y, en principio, no es posible utilizarlos en máquinas pertenecientes a arquitecturas no x86 (como ARM, la arquitectura de uso mayoritario en los dispositivos móviles).

Por ello, pensamos en una segunda alternativa: el uso de un cliente VNC web. Aunque el rendimiento es menor, podemos utilizar una única versión, que funcionará en cualquier sistema operativo y en cualquier dispositivo, incluyendo PCs, móviles, tabletas,. . . Además, el cliente VNC se integrará mucho mejor con la web de *CygnusCloud*.

Considerando que los equipos antiguos que actualmente se utilizan en la UCM tienen suficiente capacidad de procesamiento para utilizar un cliente VNC web, nos decantamos por esta segunda alternativa. En su momento, valoramos dos soluciones: el *applet Java* del proyecto TigerVNC y noVNC.

Aunque en términos generales el rendimiento del applet *Java* es satisfactorio, su desarrollo ha estado detenido desde el año 2007, y en la actualidad carece de todo tipo de soporte.

Por otra parte, noVNC es un cliente VNC relativamente nuevo, escrito íntegramente en HTML5 y *JavaScript*, que se ha integrado en un gran número de soluciones como *OpenStack*, *OpenNebula*, *CloudSigma*, *Intel MeshCentral*,. . .

Sus principales características son las siguientes:

- para utilizarlo, sólo es necesario tener instalado un navegador web razonablemente reciente.
- se adapta a los tamaños y orientaciones de pantalla de móviles y tabletas, lo que permite utilizarlo en PCs y en estos dispositivos, cada vez más extendidos.
- se distribuye bajo la licencia *Mozilla Public License* versión 2, lo que permite modificarlo cuanto sea necesario a la hora de realizar su integración con la web de *CygnusCloud*.
- soporta tráfico cifrado
- ajusta el número de colores y la resolución de la pantalla en función de la potencia del dispositivo que utilizan los usuarios.

Puesto que noVNC tiene características muy interesantes, requiere instalar menos software en los PCs en los que trabajan los usuarios y, sobre todo, cuenta con un buen soporte, decidimos utilizarlo en *CygnusCloud*.

4.4.10. Uso de la librería de virtualización `libvirt`

Como mencionamos en la sección 4.2.2, cada distribución *Linux* tiene sus propios procedimientos de configuración. Así, aunque utilicemos el mismo hipervisor, la forma de manipular las máquinas y redes virtuales puede variar de unas distribuciones a otras.

Además, interactuar directamente con el hipervisor no es nada conveniente: si se decide utilizar otro distinto, será necesario introducir muchas modificaciones.

Para evitar la aparición de estos dos problemas, hemos decidido utilizar la librería de virtualización por excelencia: `libvirt`. Sus principales características son las siguientes:

- soporta muchos sistemas de virtualización, entre los que están Xen, KVM, VirtualBox y VMWare.
- aunque está escrita en C, dispone de *bindings* para muchos otros lenguajes, como Java o Python.
- se distribuye bajo licencia LGPL, por lo que es posible utilizarla incluso desde *software* privativo.
- a través de ella, es posible crear y destruir redes virtuales, y también crear, destruir y migrar (mover de servidor) máquinas virtuales en ejecución.

Tras examinar las características de `libvirt` y evaluar nuestras necesidades, concluimos que esta librería nos proporcionaba todas las funciones que necesitábamos para manipular máquinas y redes virtuales en *CygnusCloud*.

Por otra parte, al inicio del proyecto no encontramos ninguna librería similar, por lo que la única alternativa al uso de `libvirt` era interactuar directamente con el hipervisor. Así pues, decidimos utilizar `libvirt`.

4.4.11. Uso de *Python* 2.7 como principal lenguaje de programación

Tal y como mencionamos en la sección 4.2.2, debido a las distintas características y procedimientos de configuración de las distribuciones *Linux*, es altamente recomendable utilizar lenguajes interpretados para facilitar la difusión del sistema. Los lenguajes de este tipo que consideramos al inicio del proyecto fueron esencialmente dos: *Java* y *Python*.

El uso de *Python* tenía las siguientes ventajas:

- el grado de adopción de *Python* no ha hecho más que crecer a lo largo de los últimos años, por lo que aprender a utilizarlo era muy importante para mejorar nuestras perspectivas laborales.
- a diferencia de *Java*, *Python* es *software* libre, por lo que sus restricciones de uso son considerablemente menores que las de *Java*.
- la interacción entre *scripts* y programas escritos en *Python* es claramente mejor en *Python* que en *Java*.
- los programas escritos en *Python* son mucho más breves y más fáciles de leer que los programas escritos en *Java*, lo que facilita la comprensión del código.
- para ejecutar un programa *Python* se utiliza su propio código fuente, lo que facilita enormemente la resolución de los problemas que aparecen al utilizar componentes de terceros.
- *Python* forma parte de la instalación estándar de la inmensa mayoría de distribuciones *GNU/Linux*, lo que simplifica el proceso de instalación de *CygnusCloud*.

Los grandes inconvenientes del uso de *Python* eran principalmente dos:

- era necesario aprender a utilizar un lenguaje totalmente nuevo para nosotros, cosa que no ocurría en el caso de *Java*.
- en general, hay muchas más bibliotecas y *frameworks* escritos en *Java* que escritos en *Python*, lo que limitaba las alternativas que podíamos considerar a lo largo del desarrollo del proyecto.

Ante el gran número de ventajas que tenía el uso de *Python*, estos dos inconvenientes nos parecieron aceptables, por lo que decidimos utilizar *Python* como el principal lenguaje de programación de *CygnusCloud*.

Por otra parte, en la actualidad existen dos familias de versiones de *Python*: las derivadas de la versión 2.7 y las derivadas de la versión 3.0. Ambas familias son incompatibles a nivel sintáctico, aunque es cierto que los cambios que es necesario introducir para utilizar código de la familia 2.7 en la familia 3.0 suelen ser triviales.

Ahora bien, la gran mayoría de librerías escritas en *Python*, entre las que se encuentran los *bindings* de `libvirt` y los conectores de MySQL, no han sido portadas aún a la familia 3.0, por lo que no pueden utilizarse con estas versiones. Por ello, no hemos tenido más remedio que utilizar la familia 2.7.

No obstante, hemos procurado no hacer uso de ninguna de las características de la familia 2.7 que ha desaparecido en la familia 3.0. Así, la conversión del código de *CygnusCloud* a la familia 3.0 resultará bastante sencilla.

4.4.12. Uso de la librería de red *twisted*

Puesto que *CygnusCloud* se compone de sistemas diferentes que residen en máquinas distintas, es imprescindible comunicarlos. Asimismo, como las máquinas intercambian entre sí un gran volumen de datos, es muy conveniente utilizar un servicio orientado a conexión y fiable: TCP.

Para realizar las comunicaciones *socket* a *socket*, optamos por utilizar la librería de red *twisted*, que se basa en eventos y está íntegramente escrita en *Python*. Las razones que nos llevaron a elegirla fueron las siguientes:

- *twisted* es el estándar *de facto* para comunicar programas escritos en *Python* sin necesidad de manipular *sockets* directamente, y forma parte de la instalación predeterminada de la mayoría de distribuciones *GNU/Linux* (e incluso de *Mac OS X*).
- existen libros, una ingente cantidad de documentación, foros y tutoriales con abundantes explicaciones de uso.
- se trata de una librería de código abierto, distribuida bajo licencia MIT y actualizada periódicamente.
- soporta los protocolos de transporte TCP, UDP y TCP sobre SSL, por lo que proporciona toda la funcionalidad que necesitábamos a lo largo del desarrollo del proyecto.
- soporta IP versión 6. Esto permite utilizar *CygnusCloud* en redes IP versión 4 y, a medida que las organizaciones que lo utilicen hagan la transición, también en redes IP versión 6.

El gran inconveniente de esta librería es su propia documentación. Debido a su enorme extensión, hay partes desactualizadas, partes que contradicen a otras, partes sin apenas documentar... por lo que en la práctica su uso requiere seguir el método de prueba y error.

No obstante, para garantizar el correcto funcionamiento de *CygnusCloud* en múltiples distribuciones *Linux* decidimos utilizar esta librería.

4.4.13. Uso del protocolo de transferencia de ficheros FTP

En *CygnusCloud* es necesario transferir imágenes entre el repositorio de imágenes y los servidores de máquinas virtuales. Para hacerlo, lo más adecuado es utilizar uno de los múltiples protocolos de transferencia o de compartición de ficheros existentes. En su momento, consideramos los siguientes:

- **NFS** (*Network File System*). Se trata de un protocolo muy popular en sistemas tipo UNIX. La principal ventaja de este protocolo es que, una vez establecida la conexión con la máquina remota, es posible acceder a los ficheros compartidos como si estuviesen almacenados en el sistema de ficheros de la máquina local.
- **Samba**. Se trata de una reimplementación del protocolo conocido como SMB (*Server Message Block*) o CIFS (*Common Internet File System*), utilizado para compartir ficheros y dispositivos entre máquinas *Windows*. Al igual que en el caso de NFS, tras establecer la conexión los ficheros compartidos son accesibles desde el sistema de ficheros de la máquina local.
- **FTP** (*File Transfer Protocol*). Es uno de los primeros protocolos de transferencia de ficheros que se diseñaron, y por ello también uno de los más inseguros, eficientes y fáciles de utilizar. La funcionalidad de FTP es muy básica comparada con la de NFS o Samba: sólo permite navegar por el sistema de ficheros y transferir archivos entre dos máquinas.
- **FTPS** (*File Transfer Protocol Secure*). Se trata de una extensión del protocolo FTP que cifra todo el tráfico.
- **SFTP** (*Secure File Transfer Protocol*). Aunque ambos tengan nombres parecidos, este protocolo no está relacionado con el protocolo FTP. Se trata de una extensión del protocolo SSH (*Secure Shell*) que permite transferir ficheros entre máquinas.

- **HTTP** (*HyperText Transfer Protocol*) o **HTTPS** (*HyperText Transfer Protocol Secure*). Estos protocolos, que son la base de las comunicaciones en internet, también pueden utilizarse para transferir ficheros entre máquinas. La diferencia entre ambos está en el cifrado de los datos: en el caso de HTTP, los datos viajan sin cifrar, a diferencia de lo que ocurre con HTTPS.

Puesto que *CygnusCloud* se utilizará a través de una aplicación web, decidimos utilizar los protocolos que más fácilmente pudiesen integrarse con ella. Por ello, descartamos los protocolos NFS, Samba y SFTP. De esta manera, tuvimos que elegir entre los protocolos FTP, FTPS, HTTP y HTTPS.

Tal y como dijimos en la sección 4.2.1, *CygnusCloud* debe ser eficiente en el uso de los recursos. Desde el punto de vista de la eficiencia, los protocolos FTP y FTPS son mejores que HTTP y HTTPS, ya que envían mucha menos información a través de la red, y requieren mucho menos procesamiento en las máquinas que intercambian ficheros. Por ello, descartamos los protocolos HTTP y HTTPS.

Así pues, al final tuvimos que elegir entre los protocolos FTP y FTPS. Finalmente, decidimos utilizar el protocolo FTP por ser el más eficiente y rápido de los dos.

Aunque es cierto que el protocolo FTP tiene serios problemas de seguridad, no debemos olvidar que *CygnusCloud* se utilizará a través de la red troncal de la Complutense que es, en principio, segura. No obstante, para permitir el uso de *CygnusCloud* en redes menos seguras, hemos decidido adaptar el diseño y la implementación para que sea posible utilizar el protocolo FTPS con poco esfuerzo.

4.4.14. Uso del servidor FTP `pyftpdlib`

Puesto que el repositorio de imágenes almacenará una copia de todas las imágenes que es posible utilizar en el *cluster*, lo más razonable es que esta máquina albergue un servidor FTP.

Por tanto, el repositorio de imágenes debe albergar un servidor FTP. Además, para que la interacción con el servidor FTP se simplifique, resulta muy conveniente que este esté escrito en *Python*. En su momento, consideramos dos: `pyftpdlib` y un servidor FTP construido sobre la librería *twisted*.

`pyftpdlib` es un servidor FTP ligero íntegramente escrito en *Python*. Sus principales características son las siguientes:

- es gratuito, y se distribuye bajo licencia MIT.
- es muy ligero, eficiente, estable y escalable.
- funciona con cualquier versión de *Python* entre la 2.4 y la 3.3.
- implementa, entre otras cosas, FTPS (FTP Secure, FTP Seguro), el soporte de IP versión 6, el uso nombres de fichero Unicode, múltiples protocolos de autenticación y el control del ancho de banda utilizado por el servidor FTP.
- dispone de *callbacks* para detectar múltiples eventos, tales como las conexiones y desconexiones de clientes, el inicio y la finalización de transferencia de archivos,. . .
- está muy bien documentado.

La alternativa a `pyftpdlib`, construida sobre la librería de red *twisted*, tiene las siguientes características:

- es también gratuita, y también se distribuye bajo licencia MIT.
- el servidor FTP consume muy pocos recursos.
- funciona con cualquier versión de *Python* entre la 2.4 y la 2.7.
- implementa el soporte de IP versión 6 y los protocolos de autenticación más básicos.
- también dispone de *callbacks* para detectar conexiones, desconexiones, errores de autenticación,. . .

Para decidir entre ambos, los probamos durante algunos días, tras las cuales concluimos lo siguiente:

- como viene siendo habitual en la librería de red *twisted*, la documentación del servidor FTP construido sobre *twisted* es incompleta y en muchos casos incoherente.
- el rendimiento del servidor FTP `pyftpdlib` es superior al del servidor FTP basado en la librería *twisted*.
- el servidor FTP `pyftpdlib` era mucho más fácil de integrar con el repositorio de imágenes que el servidor FTP que utiliza la librería *twisted*.

Además, dado que el servidor `pyftpdlib`

- ofrece muchas más funciones que la otra alternativa y algunas como el control del ancho de banda utilizado por el servidor FTP son muy interesantes, y que
- soporta la versión 3 de *Python*, lo que facilita la adaptación del código de *CygnusCloud* para que funcione con las nuevas versiones de *Python*.

decidimos utilizar `pyftpdlib` como servidor FTP del repositorio de imágenes.

4.4.15. Almacenamiento de las imágenes de disco en formato comprimido

Para aprovechar al máximo el espacio en disco disponible en el repositorio y reducir el ancho de banda utilizado para transferir imágenes de disco entre él y los servidores de máquinas virtuales, estas deben almacenarse y transferirse en formato comprimido.

Puesto que *CygnusCloud* debe poder implantarse con coste cero, es necesario utilizar formatos de compresión libres y gratuitos. En su momento, consideramos los formatos 7-zip, zip, tar, lzma, tar.gz, tar.bz2, tar.lzma y tar.xz.

Para decidir entre ellos, hicimos un pequeño *benchmark*, utilizando todos estos formatos para comprimir una imagen de disco de 4,1 GB. En todas las pruebas,

- utilizamos la misma máquina, con una CPU Intel Core i7 2330QM, 8 GB de memoria RAM DDR3 a 1333 MHz y un disco duro Serial-ATA 2 a 7200 RPM.
- utilizamos la misma versión del kernel Linux, la 3.2.0.
- las mediciones temporales se realizaron con la orden `time`, consideran únicamente el tiempo de CPU, omitiendo los bloqueos provocados por las operaciones de entrada/salida y por los cambios de contexto forzosos que provoca el planificador del sistema operativo.
- el uso promedio medio de la CPU, de la RAM y del disco se midieron con las herramientas `iostat` y `pidstat`, pertenecientes al paquete `sysstat`[1]. Estas herramientas obtienen la información que muestran del pseudo-sistema de ficheros `/proc`.
- se especificó un nivel de compresión medio.

Los resultados del *benchmark* aparecen en las figuras 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6.

[1] El paquete `sysstat` puede descargarse desde http://sebastien.godard.pagesperso-orange.fr/

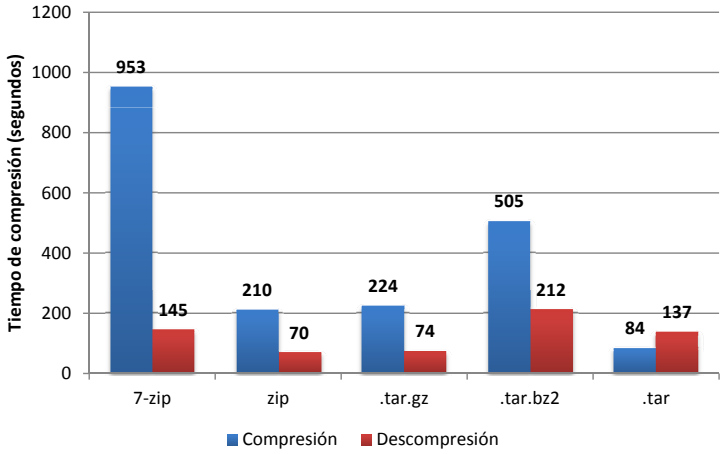

FIGURA 4.1: Tiempos de compresión y descompresión

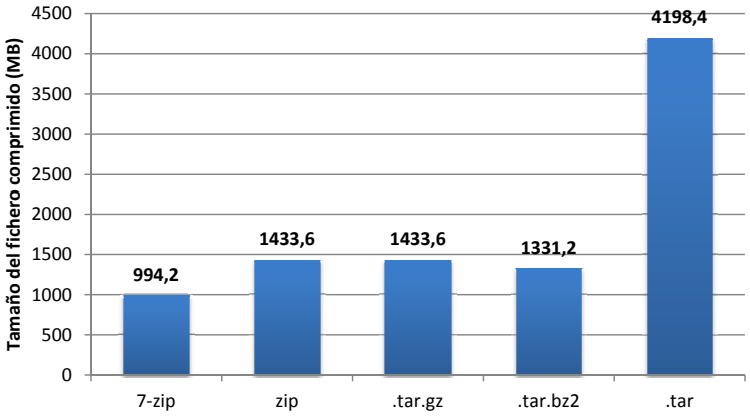

FIGURA 4.2: Tamaño del fichero comprimido

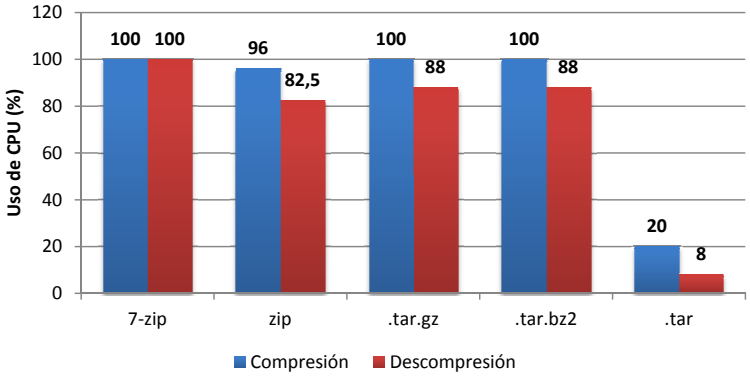

FIGURA 4.3: Uso promedio de CPU

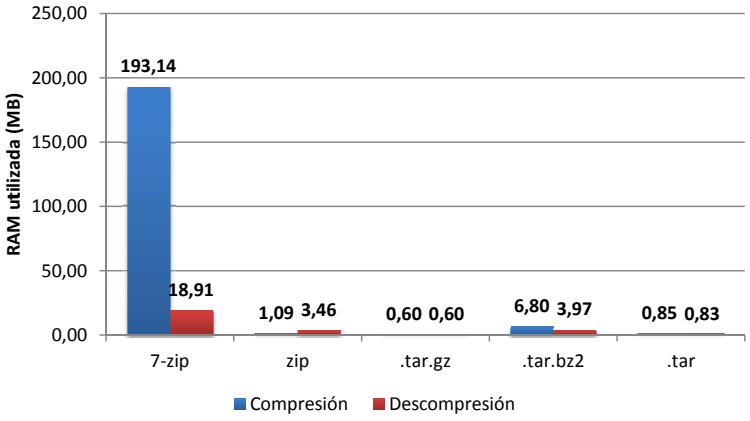

FIGURA 4.4: Uso promedio de memoria RAM

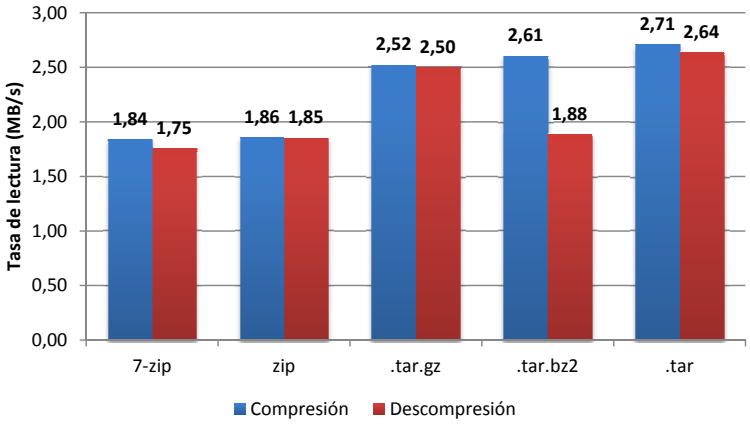

FIGURA 4.5: Tasa de lectura de disco (promedio)

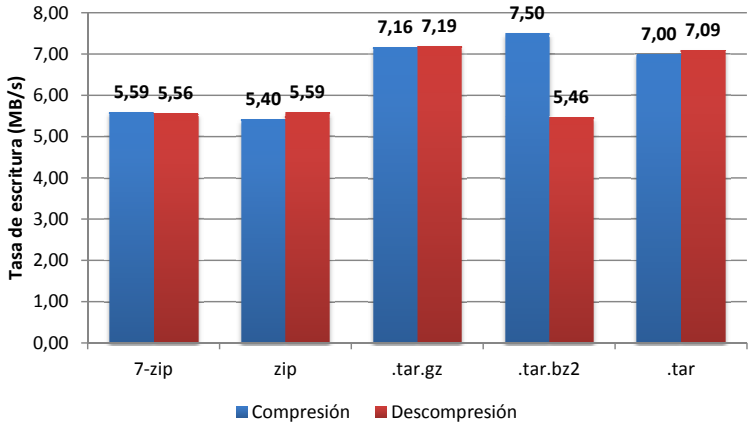

FIGURA 4.6: Tasa de escritura a disco (promedio)

Como el lector habrá observado ya, en las gráficas no aparecen los formatos lzma, tar.lzma y tar.xz. Esto se debe a que

- la compresión y la descompresión de ficheros son mucho más lentas que las del formato 7-zip,
- la tasa de compresión es muy similar a la del formato zip, y
- el uso de CPU y de memoria RAM es superior al del formato zip.

Por ello, los descartamos. Si observamos las gráficas de las figuras 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6, podemos concluir que

- el formato 7-zip consigue la tasa de compresión más alta, aunque lo hace a costa de aumentar muchísimo el tiempo de compresión y extracción, el uso promedio de CPU y el uso promedio de memoria RAM.
- los formatos zip, tar.gz y tar.bz2 alcanzan tasas de compresión muy similares. Esto se debe a que los tres usan variantes del algoritmo de Lempel-Ziv.

  No obstante, los compresores y descompresores de los dos últimos son algo más lentos que los del formato zip, utilizan más tiempo de CPU y realizan más accesos a disco.
- el formato tar alcanza la tasa de compresión más baja. Por ello, es el que menos recursos usa. Las elevadas tasas de lectura y escritura en disco se deben a que, al comprimir un fichero utilizando el formato tar, se escribe prácticamente una copia del fichero original a disco.

A la vista de estos resultados, descartamos el formato 7-zip por ser demasiado lento y consumir demasiados recursos, y el formato tar por alcanzar tasas de compresión demasiado bajas.

Así pues, tuvimos que elegir entre los formatos zip, tar.gz y tar.bz2. De todos ellos, escogimos el formato zip por las siguientes razones:

- aunque al comprimir y descomprimir se utiliza más memoria RAM que en el caso del formato tar.gz, los usos de la CPU y del disco son inferiores, por lo que los procesos de compresión afectarán menos al rendimiento global del servidor.
- es posible añadir ficheros a un fichero zip ya creado sin necesidad de descomprimirlo, lo cual aporta una gran flexibilidad. En el caso de los formatos tar.gz y tar.bz2, esto no es posible.

4.4.16. Uso de imágenes *copy-on-write*

El sistema operativo y los programas instalados en cada máquina virtual ocupan una cantidad considerable de espacio en disco. Además, el sistema operativo y las utilidades básicas no suelen ser suficientes para trabajar, por lo que frecuentemente es necesario instalar *software* adicional. Esto requiere aún más espacio en disco.

Supongamos que el sistema operativo y los programas que se van a utilizar en la máquina virtual ocupan 12 GB, que el fichero de paginación ocupa 1 GB y los usuarios disponen de 2 GB para almacenar sus datos temporales. En este caso, cada máquina virtual activa requerirá

$$12 + 1 + 2 = 15 \text{ GB de espacio en disco}$$

Como varios usuarios pueden compartir el mismo servidor de máquinas virtuales, es posible que existan varias máquinas virtuales activas de este tipo. Por ejemplo, si existen tres, serán necesarios

$$15 \cdot 3 = 45 \text{ GB de espacio libre en disco}$$

Este comportamiento presenta dos graves inconvenientes:

- las máquinas virtuales activas consumen mucho espacio en disco, lo que limitará el número de máquinas virtuales activas, y también el número de máquinas virtuales diferentes que puede albergar un mismo servidor de máquinas virtuales. Esto último no es nada conveniente para sacar el máximo partido a los servidores de máquinas virtuales.
- el arranque de una máquina virtual será muy lento. Retomando el ejemplo anterior, la máquina sólo se podrá arrancar cuando se copien los 12 GB que ocupan el sistema operativo y los programas. Esto, en un disco duro relativamente moderno, puede llevar más de cinco minutos.

Para superar estas limitaciones, sólo existen dos alternativas:

- limitar el tamaño máximo de las imágenes de disco, lo que limita también la versatilidad de *CygnusCloud*, o
- hacer que todas las máquinas virtuales activas del mismo tipo compartan la instalación del sistema operativo y de los programas. Esto es posible utilizando imágenes de disco *copy-on-write*.

Tras considerar la gran cantidad de espacio en disco que ocupan las instalaciones de *Windows*, la primera alternativa no nos pareció aceptable. Por ello, decidimos utilizar imágenes de disco *copy-on-write* para almacenar el sistema operativo y los programas instalados en las máquinas virtuales.

Así, si en el ejemplo anterior utilizásemos imágenes de disco *copy-on-write*, para alojar las tres máquinas virtuales activas necesitaríamos, aproximadamente,

$$12 + 3 \cdot (1 + 2) = 21 \text{ GB de espacio libre en disco}$$

cantidad muy inferior a la del del caso anterior.

4.4.17. Uso de dos imágenes de disco en cada máquina virtual

El uso de imágenes de disco *copy-on-write* nos permite ahorrar una gran cantidad de espacio en disco, pero también tiene un serio inconveniente: las escrituras a disco son mucho más lentas que en el caso de las imágenes de disco normales. Esta diferencia se debe al propio funcionamiento de las escrituras.

Cuando se escribe en una imagen de disco “normal”, se consulta una cabecera y se actualizan los *bytes* correspondientes del fichero. Así, las escrituras en estos ficheros son casi igual de rápidas que las escrituras directas a disco.

En cambio, cuando se escribe en una imagen de disco *copy-on-write*,

1. se calculan los cambios que introduce la escritura con respecto a la imagen original.
2. los cambios se escriben a disco, pero no en el fichero de imagen original.

Debido a esto, las escrituras en imágenes de disco *copy-on-write* son más lentas. Asimismo, también serán más lentas las posteriores lecturas de los datos escritos: para obtenerlos, hay que aplicar los cambios sobre el contenido del fichero de imagen original.

Por tanto, si nos limitamos a utilizar imágenes de disco *copy-on-write*, el rendimiento de la máquina virtual se resentirá mucho, ya que las lecturas y escrituras en el fichero de paginación son muy frecuentes, sobre todo en el caso de sistemas operativos *Windows*.

Para superar esta limitación, decidimos utilizar dos imágenes de disco en cada máquina virtual:

- una imagen del tipo *copy-on-write*, para almacenar el sistema operativo y los programas, y
- una imagen normal para almacenar el fichero de paginación y los datos temporales de los usuarios.

Tras realizar esta separación, siguen realizándose escrituras en la imagen *copy-on-write*. No obstante, estas son muy poco frecuentes, por lo que esta solución nos permite ahorrar espacio en disco y garantizar que el rendimiento de la máquina virtual no se resentirá de forma apreciable.

4.4.18. Uso del formato de imagen `qcow2`

En la sección anterior concluimos que es necesario el uso de dos imágenes de disco en cada máquina virtual: una *copy-on-write*, que almacenará los datos del sistema operativo y de las aplicaciones instaladas, y una imagen normal, que almacenará el fichero de paginación y los datos temporales de los usuarios.

Para crear las imágenes de disco *copy-on-write*, tuvimos que decidir entre dos formatos: `qcow` y `qcow2` (*QEMU Copy-On-Write* y *QEMU Copy-On-Write* 2). Sus principales características son las siguientes:

- tal y como dijimos en la sección 3.5, el hipervisor KVM utiliza los modelos de dispositivo del emulador *QEMU*. Como `qcow` y `qcow2` son los formatos de imagen nativos de *QEMU*, son también los formatos de imagen nativos de KVM.
- pueden utilizarse como imágenes de disco *copy-on-write* o como imágenes de disco normales.
- soportan compresión `zlib`.
- las imágenes de disco `qcow` y `qcow2` crecen dinámicamente a medida que se escriben datos en el disco duro de la máquina virtual. Así, el espacio libre en los discos duros virtuales no ocupará espacio en disco en los servidores de máquinas virtuales.

La principal diferencia existente entre los formatos `qcow` y `qcow2` está en las escrituras en modo *copy-on-write*. No obstante, dado que este es un aspecto de muy bajo nivel, no lo discutiremos aquí.

Como

- KVM soporta los formatos `qcow` y `qcow2`,
- resulta muy sencillo hacer conversiones entre ambos formatos,
- el formato `qcow2` es el sucesor natural del formato `qcow`, y
- con independencia de cuál de los dos formatos utilicemos, el rendimiento es prácticamente idéntico

optamos por utilizar el formato `qcow2` en las imágenes de disco *copy-on-write*.

Por otra parte, en el caso de las imágenes de disco que almacenan el fichero de paginación y los datos temporales de los usuarios consideramos los siguientes formatos:

- RAW. En realidad, no se trata de un formato de imagen, sino de un fichero que contiene directamente los *bytes* almacenados en el disco duro de la máquina virtual.

  Puesto que para acceder a disco basta con leer o escribir directamente en este fichero, esta es la alternativa que proporciona mejor rendimiento.

  No obstante, tiene un serio inconveniente: el fichero de la imagen siempre debe tener el mismo tamaño que el disco duro que representa, por lo que el espacio libre en ese disco duro ocupa espacio en disco del servidor de máquinas virtuales.

- `qcow2`. Para que el rendimiento sea adecuado, en estos casos hay que deshabilitar el modo *copy-on-write*, utilizando las imágenes de disco `qcow2` como imágenes de disco normales.

- `vmdk`. Se trata del formato nativo del sistema de virtualización VMWare, también soportado por *QEMU* (y, por tanto, por KVM).

Nuevamente, resulta muy sencillo realizar conversiones entre estos formatos. De todas formas, el hipervisor Xen no soporta el formato `vmdk`, por lo que, de cara a facilitar un hipotético cambio de hipervisor, resulta más conveniente utilizar los formatos RAW o `qcow2`.

Y puesto que al utilizar el formato RAW se está desperdiciando espacio en disco en muchos casos, también decidimos utilizar el formato `qcow2` en las imágenes de disco que almacenan el fichero de paginación y los datos temporales del usuario.

Finalmente, para que el rendimiento de las máquinas virtuales no se resienta en exceso, decidimos deshabilitar la compresión `zlib`. De esta manera, el rendimiento de la máquina virtual no se degradará de forma apreciable con respecto al uso de imágenes RAW.

4.4.19. Creación de seis familias de máquinas virtuales

Las máquinas virtuales de *CygnusCloud* pueden utilizarse con fines muy diversos. Por ejemplo, mientras que unas pueden tener instalada una *suite* ofimática y las herramientas básicas para navegar por internet, otras pueden tener instalado un entorno de desarrollo de desarrollo integrado.

En cualquier caso, todos estos usos no explotan igual la potencia del hardware de la máquina virtual. Retomando el ejemplo anterior,

- en la primera máquina, que tiene instalada la *suite* ofimática y las herramientas básicas para navegar por internet, el uso de CPU y de memoria RAM será bastante reducido. Además, se utilizará poco espacio en disco.

- en la segunda máquina, que tiene instalado el entorno de desarrollo integrado, se utilizará más memoria RAM y espacio en disco. Además, durante los procesos de compilación, el uso de CPU será muy superior al de la máquina anterior.

Además, los recursos utilizados por el sistema operativo varían de unas familias de sistemas operativos a otras. Por ejemplo,

- los sistemas operativos *Windows* de última generación requieren un mínimo de 9 GB de espacio en disco, al menos 1 GB de memoria RAM y un fichero de paginación que sea, como mínimo, un 50 % mayor que la cantidad de memoria RAM, y

- las distribuciones más recientes que utilizan el kernel *Linux* requieren, de media, unos 2 GB de espacio en disco, 512 MB de memoria RAM y un fichero de paginación que sea tan grande como la memoria RAM.

Así, desde el punto de vista del aprovechamiento de los recursos, no resulta adecuado que todas las máquinas virtuales tengan las mismas características. De hacerlo,

- se estarían desperdiciando recursos en las máquinas que tienen instalado el software menos exigente en lo que al consumo de recursos se refiere, y

- en muchos casos, no se dispondría de recursos suficientes en las máquinas que tienen instalado *software* que consume muchos recursos.

Por ello, hemos creado seis familias de máquinas virtuales. Tres de ellas están asociadas a los sistemas operativos *Windows*, y las tres restantes están asociadas a los sistemas operativos *Linux*. Sus características se recogen en los cuadros 4.1 y 4.2.

Imagen	vCPUs	RAM	Disco		Variante SO
			SO y *software*	Datos / paginación	
small	1	1 GB	20 GB	2 GB / 2 GB (4 GB en total)	32 *bits*
medium	2	2 GB	30 GB	4 GB / 4 GB (8 GB en total)	32 *bits*
big	4	3 GB	40 GB	8 GB / 8 GB (16 GB en total)	64 *bits*

CUADRO 4.1: Familias de máquinas virtuales *Windows*

Imagen	vCPUs	RAM	Disco		Variante SO
			SO y *software*	Datos / paginación	
small	1	1 GB	5 GB	2 GB / 1 GB (3 GB en total)	64 *bits*
medium	2	2 GB	10 GB	4 GB / 2 GB (6 GB en total)	64 *bits*
big	4	3 GB	15 GB	8 GB / 4 GB (12 GB en total)	64 *bits*

CUADRO 4.2: Familias de máquinas virtuales *Linux*

A la hora de fijar las características de las familias de máquinas virtuales *Windows*, hemos considerado los siguientes aspectos:

- tras desinstalar los componentes multimedia y las características del sistema operativo que no se usan con frecuencia, una instalación de *Windows* 7 ocupa 9 GB (en el caso de una versión de 32 *bits*) o 12 GB (en el caso de una versión de 64 *bits*).
- en *Windows*, se recomienda que el tamaño del fichero de paginación sea 1,5 veces el tamaño de la memoria RAM. De acuerdo a nuestra experiencia, en *Windows* se utiliza este fichero de forma intensiva, por lo que hemos decidido que su tamaño sea 2 veces el tamaño de la memoria RAM.
- las instalaciones de programas razonablemente complejos (como *Microsoft Office*) ocupan, en media, unos 2 GB de espacio en disco.
- las versiones de *Windows* de 64 *bits* utilizan, aproximadamente, el doble de memoria RAM que las de 32 *bits*. Siempre que no se utilicen las instrucciones de 64 *bits* de la CPU, cosa que en la actualidad ocurre con muy poca frecuencia, el rendimiento de todas las aplicaciones en ambas versiones es prácticamente el mismo.

  Como las familias de máquinas virtuales *Windows small* y *medium* no cuentan con 4 GB de RAM, en principio no sacarán partido a un sistema operativo de 64 *bits*, por lo que en estos casos recomendamos el uso de sistemas operativos de 32 *bits*.

  Por otra parte, aunque la familia de máquinas virtuales *Windows big* tampoco cuenta con 4 GB de memoria RAM, sí es posible que, por los recursos que requiere, el software instalado en estas máquinas saque partido a las instrucciones de 64 *bits* de la CPU del servidor de máquinas virtuales, por lo que en este caso recomendamos el uso de un sistema operativo de 64 *bits*.

  De todas formas, estas recomendaciones no tienen por qué cumplirse y, en caso de que se ignoren, el rendimiento de la infraestructura no se verá afectado.

Por otra parte, cuando fijamos las características de las familias de máquinas virtuales *Linux* consideramos los siguientes aspectos:

- en la mayoría de distribuciones Linux, cuando hay mucho software instalado se suelen utilizar unos 7 GB de espacio en disco (menos si se usa un entorno de escritorio ligero).
- las versiones de 64 *bits* de Linux usan la misma cantidad de RAM y espacio en disco que las de 32 *bits*. Además, permiten utilizar las instrucciones de 64 *bits* de la CPU del servidor de máquinas virtuales. Por ello, recomendamos el uso de versiones de *Linux* de 64 *bits*.

  Nuevamente, el rendimiento de la infraestructura no se verá afectado si se ignora esta recomendación.

Finalmente, como el lector habrá notado ya, todas las familias de máquinas virtuales tienen asignada una generosa cantidad de espacio en disco. Con esto, pretendemos incentivar el uso de máquinas virtuales con menos CPUs y menos memoria RAM, ya que estos recursos son mucho más escasos que el espacio en disco.

Además, como dijimos en la sección 4.4.18, las imágenes de disco que utilizamos crecen dinámicamente a medida que se escribe en el disco duro de la máquina virtual. Por ello, el espacio libre en los discos duros virtuales no ocupará espacio en disco en el servidor ni en el repositorio de imágenes, y la existencia de discos duros virtuales de gran tamaño infrautilizados no supone un desperdicio de recursos.

4.4.20. Uso de los servidores de edición de imágenes

La creación y edición de imágenes, entre otras cosas,

- el intercambio de ficheros de gran tamaño entre el repositorio de imágenes y un servidor de máquinas virtuales, lo que consume mucho ancho de banda, y
- la compresión y descompresión de ficheros de gran tamaño en el servidor de máquinas virtuales, lo que consume mucho tiempo de CPU y realiza muchos accesos a disco.

Por tanto, si un servidor de máquinas virtuales que se utiliza para editar imágenes alberga otras máquinas virtuales, el rendimiento de estas se degradará cuando

- se compriman y descompriman ficheros, o cuando
- se transfieran ficheros entre el servidor de máquinas virtuales y el repositorio de imágenes, ya que en estos casos el tráfico FTP, el tráfico VNC y el tráfico generado por las propias máquinas virtuales competirán por el ancho de banda del enlace que conecta el servidor de máquinas virtuales a la red de la UCM.

Así, si no tomamos medidas, un reducido número de usuarios que pueden crear o editar imágenes (los profesores y los administradores) pueden perjudicar gravemente a la gran mayoría de usuarios, que se limitan a trabajar con las máquinas virtuales ya configuradas.

Para evitar esto, hemos decidido que los procesos de creación y edición de imágenes sólo tengan lugar en cierto número de servidores de máquinas virtuales, los **servidores de edición de imágenes**.

4.4.21. Cifrado selectivo del tráfico

El tráfico que genera el sistema *CygnusCloud* puede clasificarse en dos grupos:

- tráfico generado por el protocolo de escritorio remoto VNC. Se trata del tráfico mayoritario, y permite a los usuarios manipular sus máquinas virtuales utilizando las interfaces gráficas a las que están acostumbrados.
- tráfico generado por los distintos módulos de *CygnusCloud*. Este tráfico es el minoritario, y se genera al procesar peticiones como la instanciación y destrucción de máquinas virtuales, la recopilación de estadísticas dentro de los distintos *clusters*, ...

Considerando que la red de la UCM es ya de por sí segura, no sería necesario cifrar el tráfico. No obstante, para aumentar la robustez del sistema *CygnusCloud*, todo el tráfico generado por los módulos de *CygnusCloud* se protegerá mediante cifrado SSL. Esto nos permite:

- ahorrar ancho de banda. Como el tráfico mayoritario está sin cifrar, es posible utilizar el ancho de banda para soportar más conexiones a escritorio remoto y, por tanto, dar servicio a un mayor número de usuarios.
- monitorizar fácilmente las actividades que realizan los usuarios en sus máquinas virtuales.
- aumentar la robustez del sistema frente a ataques desde la propia red de la UCM.

4.4.22. Uso del *framework* web2py

web2py es una plataforma web de código abierto (licencia GPL versión 2) que permite un ágil desarrollo de aplicaciones web seguras, gestionadas por medio de bases de datos. Esta escrito y es programable en python y contiene todos los componentes necesarios para construir aplicaciones completamente funcionales.

Al ser un framework web, web2py ofrece un rígido diseño de cara a la seguridad. Así, es capaz de resolver ciertas vulnerabilidades de forma automática siguiendo unas prácticas bien establecidas. Por ejemplo, web2py gestiona automáticamente el formateo de las entradas y salidas, aplica acciones de encriptado en los ficheros subidos y controla el área de maniobra de los desarrolladores de cara a la seguridad del sistema.

Además web2py ofrece una capa de abstracción de base de datos (DAL por su acrónimo inglés), que genera código SQL de forma transparente al modelo de gestión de datos utilizado (SQLite, MySQL, PostgreSQL, Oracle ...).

Ventajas

Las principales razones por las cuales hemos decidido utilizar web2py frente a otros frameworks web basados en python tales como Django o Grok son:

- Ofrece una estructura sencilla que permite a los usuarios aprender sobre el desarrollo web sin comprometer la funcionalidad del sistema. Por esta razón, web2py no requiere instalación ni configuración, no tiene dependencias, y expone la mayor parte de su funcionalidad a través de una interfaz de navegador web.
- Se ha mantenido estable desde el primer día, ofreciendo un diseño de arriba a abajo que asegura una total compatibilidad con respecto a aplicaciones que fueron realizadas utilizando versiones anteriores.
- Ataca de manera pro activa las cuestiones de seguridad más relevantes en las aplicaciones web modernas.
- Ofrece interfaces administrativas para simplificar la creación y gestión de las bases de datos (appAdmin) y la interacción de las diferentes vistas y controladores.
- Es ligero. Todas las librerías y código necesario para desarrollar aplicaciones ocupa en torno a 2 MB.
- Es rápido, ofreciendo una velocidad un 30 % superior que un servidor Apache medio.

4.5. Vista lógica

4.5.1. Visión general

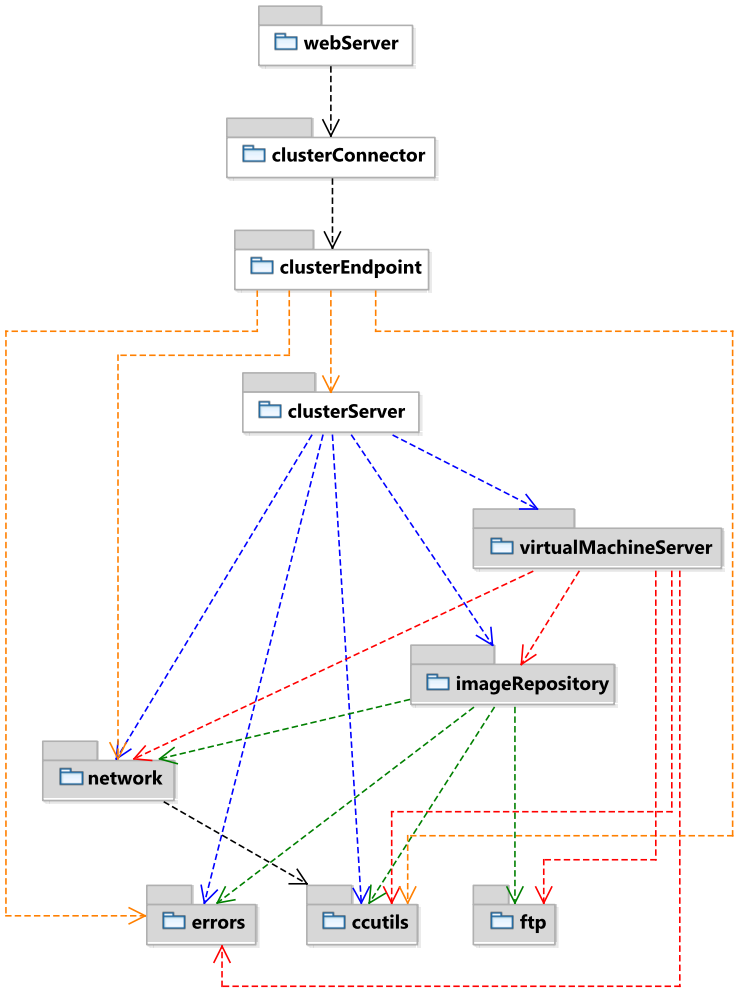

FIGURA 4.7: Diagrama de paquetes de *CygnusCloud*

La funcionalidad del sistema *CygnusCloud* está distribuida en once paquetes:

- el paquete `ccutils` contiene clases que se utilizan en varios subsistemas de *CygnusCloud*.
- el paquete `network` contiene las clases que permiten comunicar un conjunto de máquinas conectadas a la misma red. Dichas comunicaciones se realizan utilizando el protocolo TCP a través de la librería de red *twisted*.
- el paquete `virtualMachineServer` contiene las clases que implementan el subsistema servidor de máquinas virtuales.
- el paquete `clusterServer` contiene todas las clases que implementan el subsistema servidor del *cluster*.
- el paquete `imageRepository` contiene todas las clases que implementan el subsistema repositorio de imágenes.

- los paquetes `clusterEndpoint` y `clusterConnector` contienen las clases que permiten comunicar la página web de *CygnusCloud* con un servidor de *cluster*.
- el paquete `errors` contiene un tipo enumerado con todos los códigos de error generados por la infraestructura.
- el paquete `webServer` contiene las vistas y controladores asociados a la página web de *CygnusCloud*.
- el paquete `ftp` contiene el cliente y el servidor FTP utilizados para transferir imágenes de disco entre el repositorio de imágenes y los servidores de máquinas virtuales.
- finalmente, el paquete `testing` contiene clases utilizadas para depurar los sistemas servidor de máquinas virtuales y servidor del *cluster*.

Las relaciones existentes entre estos paquetes aparecen en la figura 4.7. Por claridad,

- las dependencias del paquete `virtualMachineServer` aparecen en rojo,
- las dependencias del paquete paquete `clusterServer` aparecen en azul,
- las dependencias del paquete `imageRepository` aparecen en verde,
- las dependencias del paquete `clusterConnector` aparecen en naranja, y
- el resto de dependencias entre paquetes aparecen en negro.

Finalmente, algunos de los paquetes que acabamos de presentar se descomponen en otros paquetes más pequeños. Para facilitar la visualización de las dependencias principales entre paquetes, hemos decidido omitir dichas descomposiciones. A medida que presentemos el diseño, las mostraremos junto con las relaciones entre clases que justifican las dependencias entre paquetes.

4.5.2. Paquetes y clases significativos de la arquitectura

En esta sección, mostraremos la estructura interna de los paquetes más significativos de la arquitectura. También mostraremos sus clases más relevantes, junto con una breve descripción de sus responsabilidades.

Aunque el paquete `errors` es significativo desde el punto de vista de la arquitectura, sólo contiene un tipo enumerado, en el que se definen los códigos de error. Por ello, lo omitiremos en esta sección.

4.5.2.1. `ccutils`

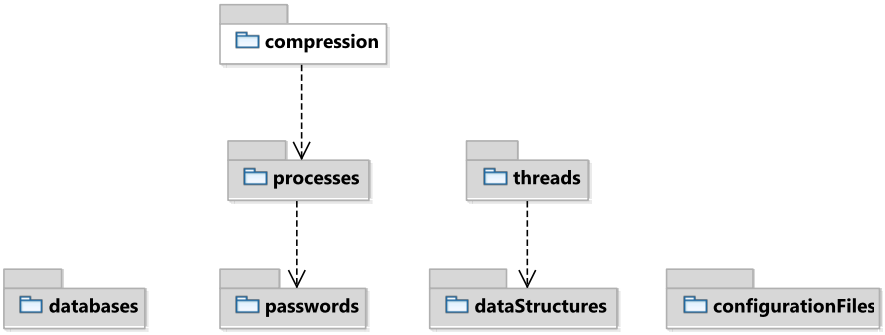

FIGURA 4.8: Descomposición del paquete `ccutils`

El paquete `ccutils` contiene clases que se utilizan en múltiples subsistemas de *CygnusCloud*. Como puede observarse en el diagrama de paquetes de la figura 4.8, el paquete `ccutils` se descompone en siete subpaquetes:

- el paquete `databases` contiene dos clases que permiten manipular una base de datos MySQL.

  La primera de ellas, `DBConfigurator`, permite crear, destruir y configurar una base de datos a través del conector no oficial de MySQL `mysqldb`.

  La segunda, `BasicDBConnector`, utiliza el conector oficial de MySQL para realizar consultas y actualizaciones sobre una base de datos. Por ello, es el antepasado común del resto de clases que manipulan una base de datos.

- el paquete `passwords` contiene una única clase, `RootPasswordHandler`, que permite obtener y recuperar la contraseña del usuario *root*.

- la principal clase del paquete `processes`, `ChildProcessManager`, permite lanzar procesos hijos en *background* y en *foreground*, bien como el usuario actual o bien como *root*.

- el paquete `dataStructures` contiene envoltorios de las principales estructuras de datos de la librería estándar de *Python*. Estos envoltorios permiten utilizar dichas estructuras de datos en entornos multihilo de forma segura.

- el paquete `threads` contiene dos clases: `BasicThread` y `QueueProcessingThread`. La primera es el antepasado común de todos los hilos de la infraestructura. Por otra parte, la segunda se corresponde con un hilo que, como su nombre indica, procesa los elementos de una cola.

- finalmente, las clases del paquete `configurationFiles` se utilizan para procesar los distintos ficheros de configuración. La más relevante de todas, `ConfigurationFileParser`, procesa los ficheros de configuración haciendo uso del módulo `ConfigParser` de la librería estándar de *Python*.

4.5.2.2. `network`

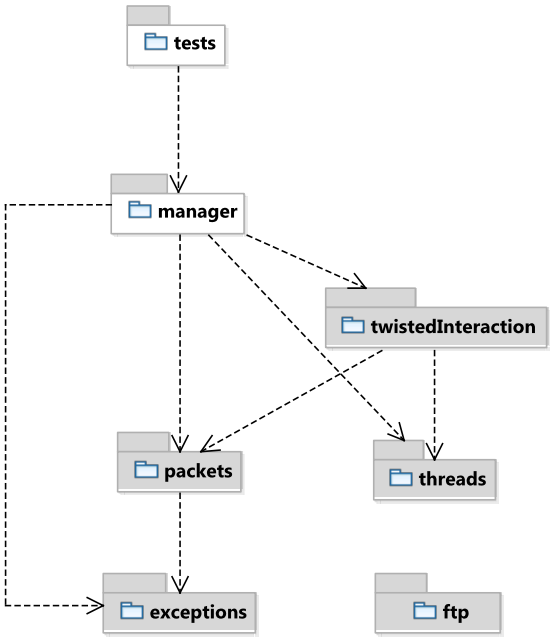

FIGURA 4.9: Descomposición del paquete `network`

Las clases del paquete `network` interactúan con la librería de red *twisted* para comunicar una máquina con otra u otras conectadas a la misma red. El diagrama de paquetes de la figura 4.9 muestra que este paquete se descompone en ocho subpaquetes:

- el paquete exceptions, que contiene todas las clases de excepción que utilizan las clases del paquete network.
- el paquete packets, que contiene la clase Packet. Esta clase se ocupa de la serialización y deserialización de la información que intercambian los equipos.
- el paquete ftp. Sus clases más importantes son las siguientes:
  - FTPClient, que permite interactuar con el cliente FTP de la librería estándar de *Python* a un elevado nivel de abstracción.
  - ConfigurableFTPServer, que ofrece una interfaz de alto nivel para interactuar con el servidor FTP pyftpdlib.
  - FTPCallback, que define la interfaz que se usará para procesar los eventos generados por el servidor FTP pyftpdlib.
- el paquete threads. Este contiene las clases ConnectionMonitoringThread, DataProcessingThread y TwistedReactorThread, correspondientes a los distintos tipos de hilo que se utilizan en la implementación de la red.
- el paquete twistedInteraction, que contiene las clases
  - Connection, ServerConnection y ClientConnection, asociadas a las conexiones de red,
  - ConnectionStatus, asociada al estado de una conexión de red, y
  - CygnusCloudProtocol y CygnusCloudProtocolFactory, utilizadas para interactuar con la librería de red *twisted* a muy bajo nivel de abstracción.
- el paquete manager, que contiene las clases NetworkManager y NetworkCallback. La primera ofrece una interfaz de alto nivel para manipular conexiones de red, y la segunda define la interfaz que se utiliza para procesar los datos recibidos a través de la red.
- el paquete tests, que contiene las pruebas de los módulos de este paquete.
- el paquete ftp. Sus principales clases son FTPClient, correspondiente a un cliente FTP basado en el de la librería estándar de *Python*, y ConfigurableFTPServer, un servidor FTP basado en pyftpdlib.

4.5.2.3. ftp

Este paquete contiene las clases del cliente y del servidor FTP que utilizan el repositorio de imágenes y los servidores de máquinas virtuales para intercambiar imágenes de disco. Sus principales clases son las siguientes:

- FTPClient. Esta clase es un envoltorio del cliente FTP incluido en la biblioteca estándar de *Python*.
- ConfigurableFTPServer. Esta clase es un envoltorio del servidor FTP pyftpdlib.

Asimismo, el paquete ftp incluye un único subpaquete, pyftpdlibInteraction. Como su nombre indica, las clases de ese paquete realizan la interacción con el servidor pyftpdlib a un bajo nivel de abstracción.

4.5.2.4. `imageRepository`

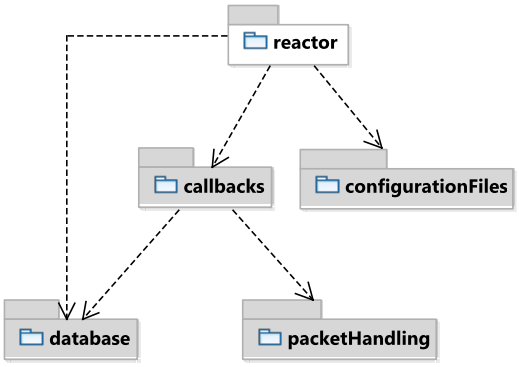

FIGURA 4.10: Descomposición del paquete `imageRepository`

Este paquete contiene las clases del subsistema repositorio de imágenes. Como puede observarse en el diagrama de paquetes de la figura 4.10, el paquete `imageRepository` se descompone en cinco subpaquetes:

- el paquete `database`. Contiene la clase `ImageRepositoryDBConnector`, que permite manipular la base de datos del repositorio de imágenes.
- el paquete `callbacks`, que contiene las interfaces `FTPServerCallback` y `CommandsCallback`. Estas se utilizarán para procesar los eventos generados por el servidor FTP pyftpdlib y para procesar los paquetes recibidos respectivamente.
- el paquete `configurationFiles`. Su única clase, `ImageRepositoryConfigurationFileParser`, parsea el fichero de configuración del demonio del repositorio de imágenes.
- el paquete `packetHandling`. Incluye una única clase, `ImageRepositoryPacketHandler`, permite crear y leer los distintos tipos de paquete que se utilizan en el repositorio de imágenes.
- el paquete `reactor`. Su única clase, `ImageRepositoryReactor`, se ocupa de realizar los procesos de inicialización y apagado del demonio del repositorio de imágenes.

4.5.2.5. `virtualMachineServer`

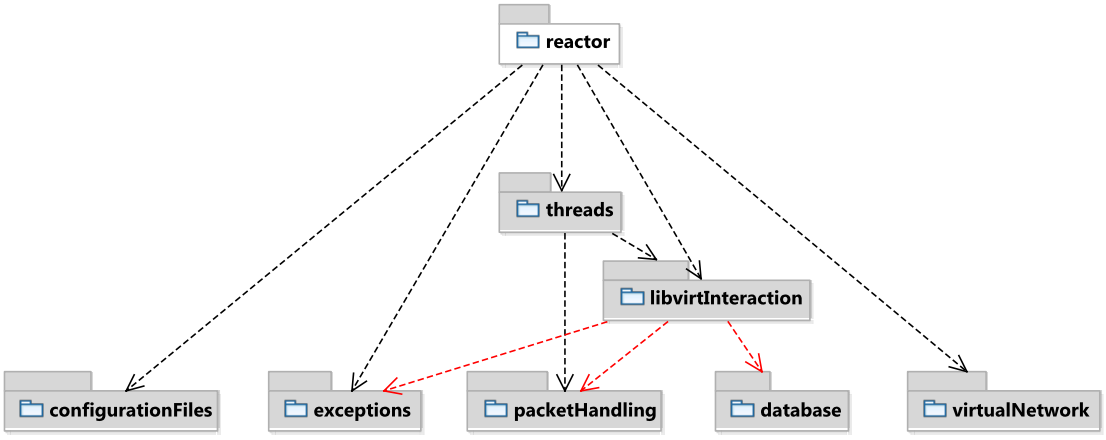

FIGURA 4.11: Descomposición del paquete `virtualMachineServer`

Este paquete contiene las clases del subsistema servidor de máquinas virtuales. El diagrama de paquetes de la figura 4.11 muestra la descomposición del paquete `virtualMachineServer`. Por claridad, las dependencias del paquete `libvirtInteraction` aparecen en rojo, y el resto de dependencias aparecen en negro.

Como puede observarse en el diagrama de paquetes de la figura 4.11, el paquete `virtualMachineServer` se descompone en siete subpaquetes:

- `configurationFiles`. La única clase que contiene, `VMServerConfigurationFileParser`, parsea el fichero de configuración del demonio del servidor de máquinas virtuales.
- `database`. Su clase más relevante, `VMServerDBConnector`, permite manipular la base de datos del servidor de máquinas virtuales.
- `exceptions`. Este paquete contiene la clase `VMServerException`, la clase de excepción asociada a la mayoría de clases del subsistema servidor de máquinas virtuales.
- `libvirtInteraction`. Este paquete contiene las clases `DomainHandler`, `LibvirtConnector` y `ConfigurationFileEditor`, que se utilizan para crear y destruir dominios interactuando con la librería `libvirt`.
- `packetHandling`. Este paquete contiene la clase `VMServerPacketHandler`, que permite crear y leer los distintos tipos de paquete asociados al servidor de máquinas virtuales.
- `reactor`. La principal clase de este paquete, `VMServerReactor`, procesa los paquetes enviados por el servidor de *cluster*.
- `threads`. Este paquete contiene las clases de hilo `CompressionThread` y `FileTransferThread`. Estas están asociadas a la compresión y descompresión de imágenes y al intercambio de imágenes con el repositorio de imágenes respectivamente.
- `virtualNetwork`. La única clase de este paquete, `VirtualNetworkManager`, dispone de métodos para crear y destruir redes virtuales.

4.5.2.6. clusterServer

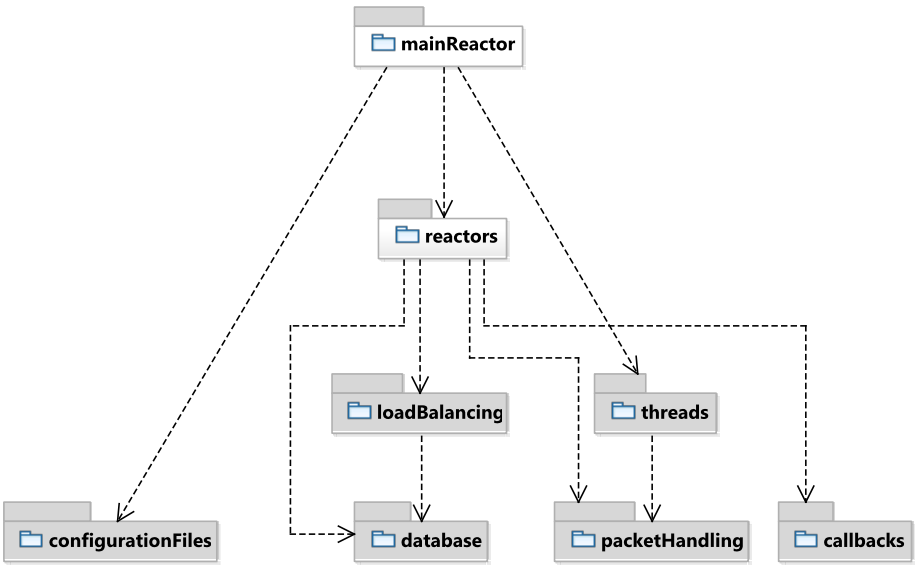

FIGURA 4.12: Descomposición del paquete `clusterServer`

Tal y como se muestra en el diagrama de paquetes de la figura 4.12, el paquete `clusterServer` se descompone en ocho subpaquetes:

- `configurationFiles`. Nuevamente, este paquete contiene la clase que procesa el fichero de configuración del demonio del servidor de *cluster*, `ClusterServerConfigurationFileParser`.
- `callbacks`. En él se definen las interfaces `ClusterEndpointCallback`, `ImageRepositoryCallback` y `VMServerCallback`, que se usarán para procesar los paquetes recibidos desde el servidro web, el repositorio de imágenes y los servidores de máquinas virtuales respectivamente.
- `database`. Su clase más importante, `ClusterServerDBConnector`, permite manipular la base de datos del servidor de *cluster*.
- `loadBalancing`. Contiene la interfaz `LoadBalancer`, común a todos los algoritmos de balanceado de carga, y la clase `PenaltyBasedLoadBalancer`, asociada al algoritmo de balanceado de carga basado en penalizaciones.
- `mainReactor`. Contiene la clase `ClusterServerMainReactor`, el punto de entrada del demonio del servidor de *cluster*.
- `packetHandling`. Contiene la clase `ClusterServerPacketHandler`, que permite manipular los paquetes que intercambian el servidor de *cluster* y el servidor web.
- `reactors`. Contiene las clases `EndpointPacketReactor`, `ImageRepositoryPacketReactor`, `VMServerPacketReactor` y `NetworkEventsReactor`. Las tres primeras procesan los paquetes recibidos desde el servidor web, el repositorio de imágenes y el servidor de máquinas virtuales respectivamente. La última procesa los eventos de desconexión y reconexión generados por la red.
- `threads`. Su única clase, `ClusterStatusMonitoringThread`, se corresponde con un hilo que envía periódicamente las actualizaciones de estado a todas las máquinas del *cluster*.

4.5.2.7. `clusterEndpoint`

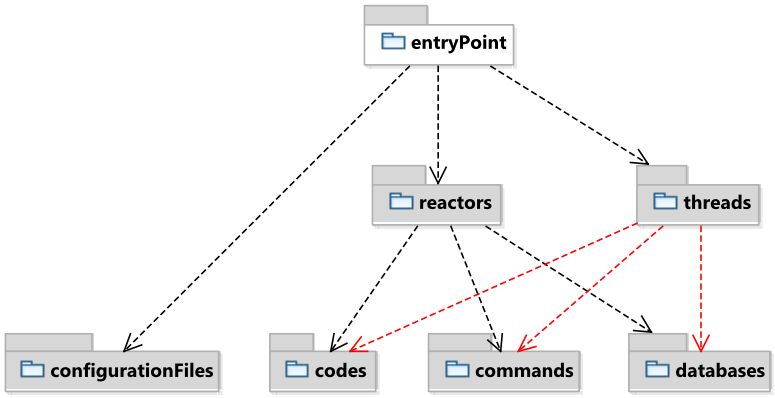

FIGURA 4.13: Descomposición del paquete `clusterEndpoint`

El diagrama de paquetes de la figura muestra la descomposición del paquete `clusterEndpoint`. Como se puede apreciar en él, el paquete `clusterEndpoint` se descompone en siete subpaquetes:

- `codes`. Este paquete contiene la interfaz `CodesTranslator`, que se utilizará para convertir a *strings* los códigos de error generados por la infraestructura, y la clase `SpanishCodesTranslator`, que genera esos *strings* en castellano.
- `commands`. Este paquete contiene la clase `CommandsHandler`, que realiza la serialización y deserialización de los comandos y de las salidas de comandos que usa la página web para interactuar con la infraestructura.

- configurationFiles. Este paquete contiene la clase ClusterEndpointConfigurationFileParser, que manipula el fichero de configuración del demonio que conecta la página web con la infraestructura.
- databases. Sus principales clases son CommandsDBConnector y ClusterEndpointDBConnector, que permiten manipular las dos bases de datos que utiliza el servidor web.
- entryPoint. Su única clase, ClusterEndpointEntryPoint, es el punto de entrada del demonio que conecta la página web con la infraestructura.
- reactors. Contiene las clases CommandsProcessor y PacketReactor, que se usan para procesar las órdenes enviadas a través de la página web y los paquetes enviados desde el servidor de *cluster* respectivamente.
- threads. Contiene las clases de hilo CommandsMonitoringThread y DatabaseUpdateThread, que se utilizan para detectar *timeouts* en la ejecución de los comandos que envía la página web y para recopilar periódicamente el estado de la infraestructura respectivamente.

4.5.2.8. clusterConnector

El paquete clusterConnector contiene una única clase, ClusterConnector, que permite manipular las bases de datos del servidor web y enviar peticiones a la infraestructura desde la página web.

4.5.2.9. testing

El paquete testing contiene clases que permiten comprobar el correcto funcionamiento de los demonios del repositorio de imágenes, del servidor de máquinas virtuales y del servidor de *cluster*. Estas clases son las siguientes:

- ImageRepositoryTester. Esta clase interactúa con el demonio del repositorio de imágenes tal y como lo harían el demonio del servidor de máquinas virtuales y el demonio del servidor de *cluster*. Por ello, permite comprobar el correcto funcionamiento de todas las funciones del repositorio de imágenes.
- VirtualMachineServerTester. Esta clase interactúa con el demonio del servidor de máquinas virtuales tal y como lo haría el demonio del servidor de *cluster*. Así pues, permite comprobar el correcto funcionamiento de todas las funciones del servidor de máquinas virtuales.
- ClusterServerTester. Esta clase interactúa con el demonio del servidor de *cluster* tal y como lo haría el servidor web. Esto permite comprobar el correcto funcionamiento de todas las funciones del servidor de *cluster*.

4.5.2.10. webServer

El paquete webServer se encuentra distribuido en un conjunto de paquetes que hacen referencia a los diferentes tipos de elementos necesarios para implementar la funcionalidad de la web. Estos paquetes, que se muestran en el diagrama de paquetes de la figura 4.14, son:

- Views. Paquete con el conjunto de vistas que definen el aspecto y la estructura de las páginas.
- Controllers. Paquete que contiene los módulos python encargados de manejar las diferentes funcionalidades de la web.
- Models. Incluye dos módulos python encargados de definir dos aspectos importantes de la web. El primer módulo, menu, introduce la información sobre la versión , las palabras clave y todos los aspectos generales de la web. El segundo, db, se encarga de crear las tablas de la base de datos que serán utilizadas para almacenar la información manejada por la web.
- Languages. Incluye un conjunto de diccionarios que permiten traducir la web a diferentes idiomas.

- `Static`. Este paquete incluye los ficheros `.css`, `.js` y las imágenes necesarias para definir el aspecto de la web. También incluye el controlador noVNC y todos los ficheros `.html` auxiliares que se integran en la web.
- `Modules`. En este paquete se encuentran todos los módulos *Python* auxiliares que serán utilizados por los controladores.

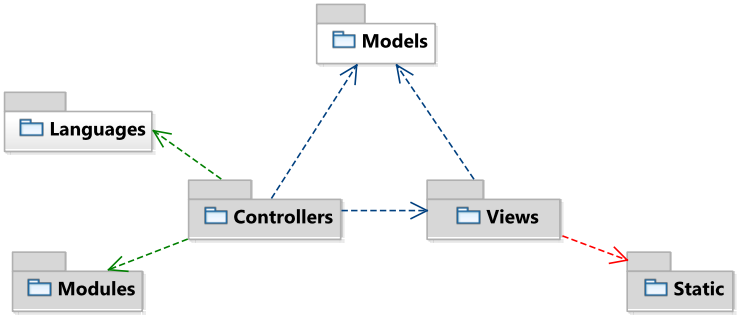

FIGURA 4.14: Diagrama de paquetes de la web

4.5.3. El paquete `network`

Los módulos de este paquete proporcionan una interfaz que permite utilizar la librería de red `twisted` a un alto nivel de abstracción. Puesto que el diseño de esta parte del sistema está íntimamente relacionado con el funcionamiento de la librería de red `twisted`, es fundamental que el lector esté familiarizado con los conceptos básicos de esta librería.

Por ello, comenzaremos mostrando, en líneas generales, cómo funciona la librería de red `twisted`. Posteriormente, describiremos el contenido de este paquete en orden creciente del nivel de abstracción, profundizando más en el funcionamiento de `twisted` cuando sea preciso.

En cualquier caso, dado que `twisted` es una librería con muchas funciones, sólo mostraremos los aspectos de su funcionamiento más relacionados con el diseño del paquete `network`. Si el lector está interesado en ampliar la información que aquí le proporcionamos, le remitimos a la documentación oficial ([13]) y al libro *Twisted Network Programming Essentials* ([14]).

4.5.3.1. Versión del protocolo IP utilizada

Como dijimos en la sección 4.4.4, la red troncal de la Complutense sigue utilizando el protocolo IP versión 4.

Así, para realizar una prueba piloto o implantar *CygnusCloud*, será necesario que todas las máquinas de la infraestructura utilicen la versión 4 de dicho protocolo en todas sus comunicaciones.

De ahora en adelante, siempre que hablemos de direcciones IP o de datagramas IP nos estaremos refiriendo a direcciones IP versión 4 y a datagramas IP versión 4.

Asimismo, también asumiremos que todos los protocolos de la capa de transporte (como TCP y UDP) y todos los protocolos de la capa de aplicación que los utilizan se apoyan en el protocolo IP versión 4 para transferir y recibir datos por la red.

4.5.3.2. La librería de red `twisted`: visión general

Tal y como adelantamos en la sección 4.4.12, `twisted` es una librería de red basada en eventos. Así, muchas funciones que realiza, como la recepción de datos y el establecimiento de conexiones son totalmente asíncronas.

Para generar los eventos, `twisted` utiliza el patrón *reactor*, es decir, muestrea periódicamente el estado de los *sockets* y actúa en consecuencia. De esta manera, todas las aplicaciones que utilicen `twisted` seguirán el siguiente esquema de comportamiento:

1. el código de la librería muestrea los *sockets* abiertos por la aplicación para detectar cambios.
2. cuando se detecta un cambio, se genera un evento para informar del cambio al código del cliente.
3. el código del cliente utiliza la información asociada al evento y procesa el cambio.
4. cuando termina, el código del cliente devuelve el control al código de la librería.

Por otra parte, el envío de información a través de la red es, como cabría esperar, síncrono.

Finalmente, el muestreo de los *sockets* y el tratamiento de los cambios tiene lugar en un bucle que se ejecuta permanentemente: el *bucle reactor*.

4.5.3.2.1. Protocolos y factorías de protocolos

La forma en que se procesan los datos recibidos a través de la red varía de unas aplicaciones a otras, e incluso puede variar en una misma aplicación.

Por ejemplo, en *CygnusCloud* algunos datos recibidos se escriben en una base de datos, otros se reenvían y otros se le muestran directamente al usuario.

Por ello, es necesario que el proceso de generación y tratamiento del evento no dependa de los datos concretos que manipula cada aplicación. Para hacer esto posible, *twisted* define los *protocolos* y las *factorías de protocolos*.

Un *protocolo* es un objeto que trata eventos de red (como la recepción de un segmento TCP) y que puede enviar datos al otro extremo de la conexión. Por su parte, las *factorías de protocolos* son, como su nombre indica, objetos capaces de instanciar y configurar protocolos.

Los protocolos y las factorías de protocolos se manipulan siempre a través de dos interfaces, que se definen en las clases abstractas `Protocol` y `Factory` respectivamente.

4.5.3.2.2. El bucle reactor

Su misión principal es muestrear periódicamente todos los *sockets* abiertos e invocar a las rutinas de tratamiento adecuados cuando se produzca un evento. Como ya hemos dicho, las rutinas de tratamiento se definen en subclases concretas de `Protocol`.

En la librería *twisted* se definen varias variantes del bucle reactor, que se diferencian fundamentalmente por la forma de realizar el muestreo periódico. Nosotros utilizamos el que está definido en la clase `PollReactor`, que se limita a muestrear periódicamente todos los *sockets* para determinar si se han producido cambios.

Por otra parte, para que el funcionamiento de la librería sea el correcto, deben cumplirse dos restricciones:

- aunque en cada aplicación puede haber un número arbitrario de conexiones activas, sólo puede existir *un* único bucle reactor.
- una vez que se detiene el bucle reactor, este *no* puede reiniciarse mientras la aplicación se siga ejecutando. Por ello, sólo se puede salir de este bucle cuando la aplicación no va a utilizar más la red.

Finalmente, los eventos de red se tratan en el propio bucle del reactor. Para garantizar un tiempo de respuesta adecuado, es imprescindible que los métodos de tratamiento sean tan rápidos como sea posible y que, en la medida de lo posible, no produzcan bloqueos.

4.5.3.2.3. Establecimiento de conexiones

Para establecer una conexión, es necesario determinar

- el protocolo de transporte que esta empleará (como TCP o UDP), y

- el papel que desempeña la máquina en la conexión (cliente o servidor).

Para eliminar en la medida de lo posible los detalles de bajo nivel de nuestro código, hemos utilizado una función que se ha añadido recientemente a la librería *twisted*: los *endpoints*.

Un *endpoint* es un objeto que sirve para configurar uno de los extremos de la conexión. Todos los protocolos de transporte implementados en *twisted* definen clases de *endpoints*: una para la máquina servidor y otra para las máquinas cliente. En el proceso de conexión el objeto *endpoint*, que es una instancia de una subclase concreta de `Endpoint`,

1. registra una solicitud de conexión en el bucle reactor.
2. registra la factoría de protocolos en el bucle reactor. Cuando la conexión se cree, este objeto se utilizará para crear el protocolo asociado a la misma.

Las relaciones existentes entre la clase `Endpoint` y el resto de clases principales de la librería *twisted* aparecen en el diagrama de clases de la figura 4.15. Por claridad, en el diagrama sólo aparecen las subclases de *endpoint* que utilizamos en el paquete `network`, correspondientes a los protocolos TCP y TCP sobre SSL.

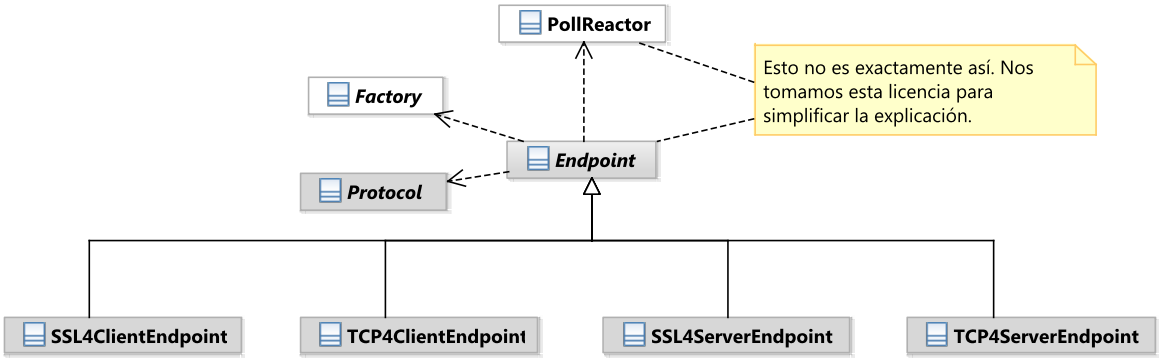

FIGURA 4.15: Relaciones entre protocolos, factorías de protocolos y *endpoints*

Por otra parte, los *endpoints* no manipulan directamente un objeto `PollReactor` durante el proceso de creación de la conexión: para ello, utilizan las interfaces que esta clase implementa. Nos hemos tomado esta licencia para simplificar la explicación.

Como ya hemos visto, los protocolos se crean al establecerse la conexión. Además, puesto que siempre están ligados a una conexión, estos objetos se destruyen cuando esta se cierra.

Finalmente, los protocolos se usan de forma distinta en función del tipo de conexión:

- en el caso de una conexión de tipo cliente existirá un único protocolo, que permitirá la comunicación bidireccional con el servidor.
- en el caso de una conexión de tipo servidor existirán tantos protocolos como clientes. Cada uno de ellos permitirá la comunicación bidireccional entre el servidor y un único cliente.

4.5.3.2.4. Envío de datos

Para enviar datos a través de una conexión se utilizan instancias de subclases concretas de `Protocol`. El reactor nunca interviene a la hora de enviar datos: a través del protocolo, los *bytes* a enviar se escriben directamente en el *socket* asociado a la conexión.

En principio, la librería *twisted* sólo impone una limitación: sólo es posible enviar *strings*, es decir, secuencias de *bytes*. Por ello, hay que serializar toda la información antes de enviarla y deserializarla al recibirla.

Es importante notar que el número de *bytes* a enviar no está limitado. Por tanto, los datos se fragmentarán en varios segmentos TCP cuando sea preciso.

4.5.3.3. La clase `Packet`

Como ya hemos mencionado, el principal objetivo de las clases del paquete `network` es proporcionar una forma de utilizar la librería de red twisted a un elevado nivel de abstracción. Para ello es necesario, como mínimo,

- hacer totalmente transparentes al usuario los procesos de serialización y de deserialización de la información que se envía y recibe, y
- fijar la prioridad de los datos que circulan por la red. Esto es fundamental para que el sistema pueda responder rápidamente ante eventos de suma importancia como la caída de un servidor de máquinas virtuales.

La clase `Packet` cubre estas dos necesidades. Sus instancias representan paquetes de red, y tienen asociada la siguiente información:

- un **tipo**. Existen dos: uno para transportar datos de gestión de la red y otro para transportar datos del cliente. Los paquetes de gestión de la red son siempre más prioritarios que los que transportan datos del cliente.
- una **prioridad**. Se trata de un valor entero, positivo en el caso de los paquetes que contienen datos de los usuarios y negativo en el caso de los paquetes de gestión de la red. Cuanto *menor* es la prioridad de un paquete, *más* prioritario será.
- una secuencia de *bytes* con los **datos** del paquete.

Tipo	Rango de la prioridad	Tamaño máximo (paquete completo)
Datos del cliente	$[0, 32767]$	64 KB
Gestión de la red	$[-32768, -1]$	64 KB

CUADRO 4.3: Características de los distintos tipos de paquete

El cuadro 4.3 recoge las características de los dos tipos de paquete. En ambos casos, el tamaño máximo del paquete completo (es decir, incluyendo la cabecera con el tipo y la prioridad y los datos) no puede exceder los 64 KB. Esto nos permite

- garantizar que no haya usuarios que acaparen el ancho de banda de la red mediante el envío de paquetes de gran tamaño.
- mantener el tiempo de respuesta de la red dentro de unos márgenes razonables. Puesto que los paquetes de pequeño tamaño tardan menos en enviarse y en ser recibidos, los paquetes más prioritarios podrán "adelantar" a los paquetes menos prioritarios que les preceden y llegar a su destino tan rápido como sea posible.
- reducir el tiempo de procesamiento de los paquetes que se van a redirigir.

4.5.3.3.1. Serialización y deserialización de los datos

Para hacer la serialización y deserialización de los datos totalmente transparentes a los usuarios, la clase `Packet` dispone de métodos de lectura y escritura de *strings*, valores enteros, valores booleanos y números en punto flotante.

Tal y como mencionamos en la sección 4.5.3.2, la librería `twisted` sólo es capaz de enviar y recibir *strings*, por lo que

- los métodos de escritura convierten a *strings* los valores a escribir en el paquete y los añaden al final de una estructura de datos intermedia, y

- los métodos de lectura extraen *strings* de la estructura de datos intermedia y los convierten en valores del tipo que el usuario quiere leer. Todas las lecturas son destructivas, es decir, cada dato escrito en el paquete sólo puede leerse una vez.

La estructura de datos intermedia es un *string*. Para leer y escribir en ella se usan los métodos de extracción de subcadenas y concatenación definidos en la librería estándar de *Python*. El formato de los datos serializados viene dado por la siguiente expresión regular:

```
(etiqueta de tipo$valor$)*
```

Los valores se serializan y deserializan con los métodos definidos en la librería estándar de *Python*. El cuadro 4.4 contiene la codificación de las etiquetas de tipo de los paquetes.

Etiqueta	Tipo de datos
0	entero
1	entero largo
2	*string*
3	número de punto flotante

CUADRO 4.4: Etiquetas de tipo que pueden aparecer en un paquete

También es necesario serializar la cabecera del paquete. Su formato es el siguiente:

```
tipo,prioridad
```

Nuevamente, tanto el tipo como la prioridad del paquete se serializan y deserializan utilizando métodos de la librería estándar de *Python*. El paquete serializado se obtiene concatenando la cabecera y los datos serializados, y su formato viene descrito por esta expresión regular:

```
tipo,prioridad(etiqueta de tipo$valor$)*
```

Finalmente, los métodos de la clase `Packet` también garantizan que los paquetes siempre están bien formados, es decir, que

- la prioridad es la adecuada para el tipo de paquete
- no se excede el tamaño máximo del paquete al escribir datos en él
- los datos se leen del paquete en el mismo orden en que se escriben

4.5.3.4. Envío y recepción de datos

Como ya mencionamos en la sección 4.5.3.2, para poder enviar y recibir información utilizando la librería `twisted` es necesario definir dos clases:

- una subclase de `Protocol`, `CygnusCloudProtocol`. Sus instancias procesarán los datos recibidos a través de cada conexión.
- una subclase de `Factory`, `CygnusCloudProtocolFactory`. Sus instancias crearán objetos CygnusCloudProtocol cuando se establezcan las conexiones.

Las relaciones entre estas clases, la clase `Packet` y las clases de la librería `twisted` aparecen en el diagrama de clases de la figura 4.16. En dicho diagrama, hemos abreviado `CygnusCloud` utilizando las siglas `CC`.

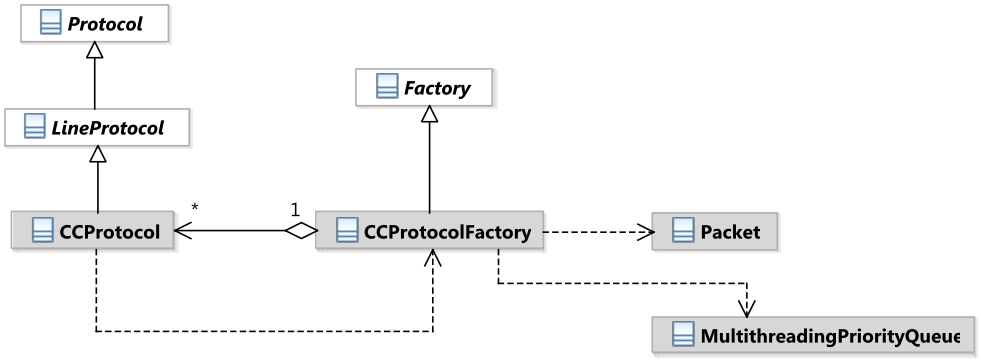

FIGURA 4.16: Interacción con *twisted* a muy bajo nivel de abstracción: diagrama de clases

En primer lugar, la clase `CygnusCloudProtocol` no hereda directamente de la clase `Protocol`, sino que lo hace a través de la clase `LineProtocol`. Esto se debe a que los datos de los paquetes pueden distribuirse en segmentos TCP de cualquier forma, por lo que, si utilizamos directamente un objeto `Protocol`, podrán producirse errores al deserializar los paquetes en el destino.

En cambio, la clase `LineProtocol`, que hereda de `Protocol`, está preparada para enviar líneas de texto a través de la conexión, característica que hemos aprovechado para extraer los paquetes de los datos recibidos.

Por otra parte, existe una dependencia cíclica entre las clases `CygnusCloudProtocol` y `CygnusCloudProtocolFactory`. Dicha dependencia está provocada por el funcionamiento de la librería `twisted`: cada par de máquinas conectadas se comunica siempre a través de una conexión bidireccional. Por ello, si n clientes se conectan a un mismo servidor,

- en los clientes habrá un único objeto `CygnusCloudProtocol`, y
- en el servidor habrá n objetos `CygnusCloudProtocol`, cada uno de los cuales permite al servidor comunicarse con un cliente concreto.

Para simplificar la implementación de la red nos interesa que, en los niveles de mayor nivel de abstracción, sólo sea necesario realizar la distinción entre clientes y servidores a la hora de establecer las conexiones.

Así, lo más conveniente es extender la funcionalidad de la clase `CygnusCloudProtocolFactory` para que, además de instanciar objetos `CygnusCloudProtocol`, sea capaz de:

- mantener una lista de referencias a todos los objetos `CygnusCloudProtocol` asociados a la conexión y actualizarla a medida que los clientes se conecten y desconecten.
- indicar si la conexión se puede utilizar, teniendo en cuenta que una conexión no está lista cuando no tiene asociado ningún objeto `CygnusCloudProtocol`.
- serializar y enviar todos los paquetes del servidor
  - a todos los clientes conectados (envío *multicast*), o
  - a uno de los clientes conectados (envío *unicast*)

  según lo que especifique el código cliente. Para ello, se utilizarán las referencias a los objetos `CygnusCloudProtocol`. En caso de que no haya ninguna, los datos a enviar se descartarán.
- procesar los datos recibidos a través de cualquier objeto `CygnusCloudProtocol`, deserializándolos para formar los paquetes y avisando a la capa superior.

Finalmente, como ya hemos dicho en repetidas ocasiones, la recepción de paquetes es totalmente asíncrona. Por motivos de eficiencia y para respetar el tipo y la prioridad de los paquetes, los objetos `CygnusCloudProtocolFactory` insertan todos los paquetes que reciben en una cola de prioridad, que es una instancia de la clase `MultithreadingPriorityQueue`. La capa superior extraerá de ella los paquetes de acuerdo a su tipo y prioridad.

4.5.3.5. Conexiones de red

Con lo que hemos mostrado hasta ahora, ya podemos interactuar con `twisted` para enviar y recibir paquetes. Pero antes es necesario establecer la conexión de red y también gestionar los recursos que tiene asociados, entre los están, en principio,

- su estado, es decir, si la conexión se está estableciendo, si ya se puede utilizar, si se está intentando restablecer, etcétera.
- la cola de paquetes recibidos, de la que hablamos en la última sección.
- la factoría de protocolos, que es, como vimos, una instancia de `CygnusCloudProtocolFactory`.
- una dirección IP, un puerto y el protocolo a utilizar (TCP o TCP sobre SSL). Esta información sólo se utiliza para establecerla.

Puesto que todos estos recursos están muy relacionados, lo más conveniente es manipularlos de forma conjunta. Esta es la finalidad de la clase `NetworkConnection` y de sus dos subclases concretas: `ClientConnection` y `ServerConnection`.

La clase abstracta `NetworkConnection` define la interfaz que utilizará la capa superior para interactuar con una conexión de red. Sus dos subclases concretas, `ClientConnection` y `ServerConnection`, se corresponden con los dos tipos de conexiones que es posible establecer: las conexiones de tipo cliente y de tipo servidor respectivamente. Estas dos clases comparten la mayor parte del código, y sólo difieren a la hora de establecer la conexión y al actualizar su estado.

El diagrama de clases de la figura 4.17 recoge las relaciones más relevantes en las que intervienen estas tres clases. A lo largo de esta sección, las explicaremos en detalle.

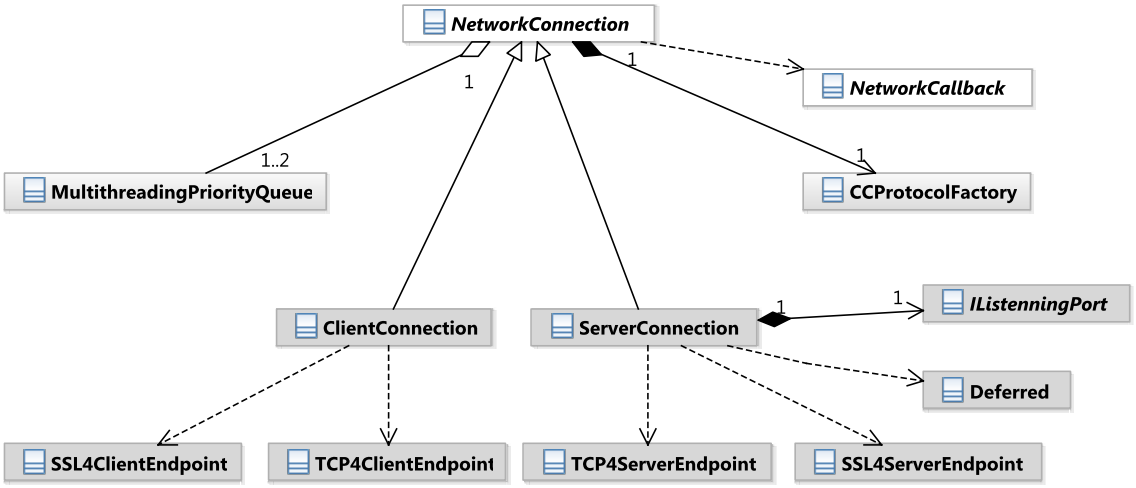

FIGURA 4.17: Relaciones más relevantes en las que intervienen las clases `NetworkConnection`, `ClientConnection` y `ServerConnection`

4.5.3.5.1. Establecimiento de una conexión

Para establecer una conexión de tipo cliente, basta con escoger un protocolo y el *endpoint* de tipo cliente que tiene asociado. Como nosotros sólo utilizamos los protocolos TCP y TCP sobre SSL, estos *endpoints* serán `TCP4ClientEndpoint` y `SSL4ClientEndpoint`.

En la sección 4.5.3.2 mostramos el intercambio de mensajes que tiene lugar durante el proceso de conexión. Por ahora, lo único que nos interesa del mismo es que, cuando el método del *endpoint* correspondiente termina, la conexión ya está lista para ser utilizada.

Por otra parte, las conexiones de tipo servidor se establecen de forma similar, pero utilizando los *endpoints* `TCP4ServerEndpoint` y `SSL4ServerEndpoint`. Pero a diferencia del caso anterior, cuando el método del *endpoint* termina la conexión aún no está lista para ser utilizada: sólo lo estará cuando se conecte el primer cliente. Esto justifica el hecho de que el estado evolucione de forma distinta en conexiones de tipo cliente y en conexiones de tipo servidor.

Finalmente, los métodos de establecimiento de una conexión siempre devuelven un objeto `Deferred`, que permite tratar los errores y, en el caso de conexiones de tipo servidor, obtener algunos de sus recursos asociados (un objeto `IListenningPort`) y cancelar el establecimiento de las mismas.

4.5.3.5.2. Envío y recepción de datos

La clase `NetworkConnection` envía paquetes a través de su objeto `CygnusCloudProtocolFactory`, cuyos métodos permiten transmitir el paquete a enviar. No obstante, los paquetes no se envían de forma instantánea, y sólo se pueden enviar uno por uno. Para que la capa superior pueda ignorar esta restricción, cada conexión también tiene asociada una cola de prioridad que contiene los paquetes a enviar.

Esa cola será una instancia de la clase `MultithreadingPriorityQueue`. Los paquetes a enviar siempre se insertan en esta cola, y se transmitirán uno por uno y cuando llegue su turno.

Por otra parte, también es necesario procesar los paquetes que el objeto `CygnusCloudProtocolFactory` deposita en la cola de paquetes recibidos. Puesto que el contenido de los paquetes que contienen datos del cliente depende del dominio de la aplicación, estos deben procesarse en el código del cliente, al que se invocará utilizando la interfaz definida por la clase abstracta `NetworkCallback`. Naturalmente, estos paquetes se borran tras ser procesados.

4.5.3.5.3. Estado de las conexiones

Dependiendo del tipo de conexión, el estado evoluciona de forma distinta. Los diagramas de las figuras 4.18 y 4.19 muestran la evolución del estado de las conexiones de tipo servidor y de tipo cliente respectivamente.

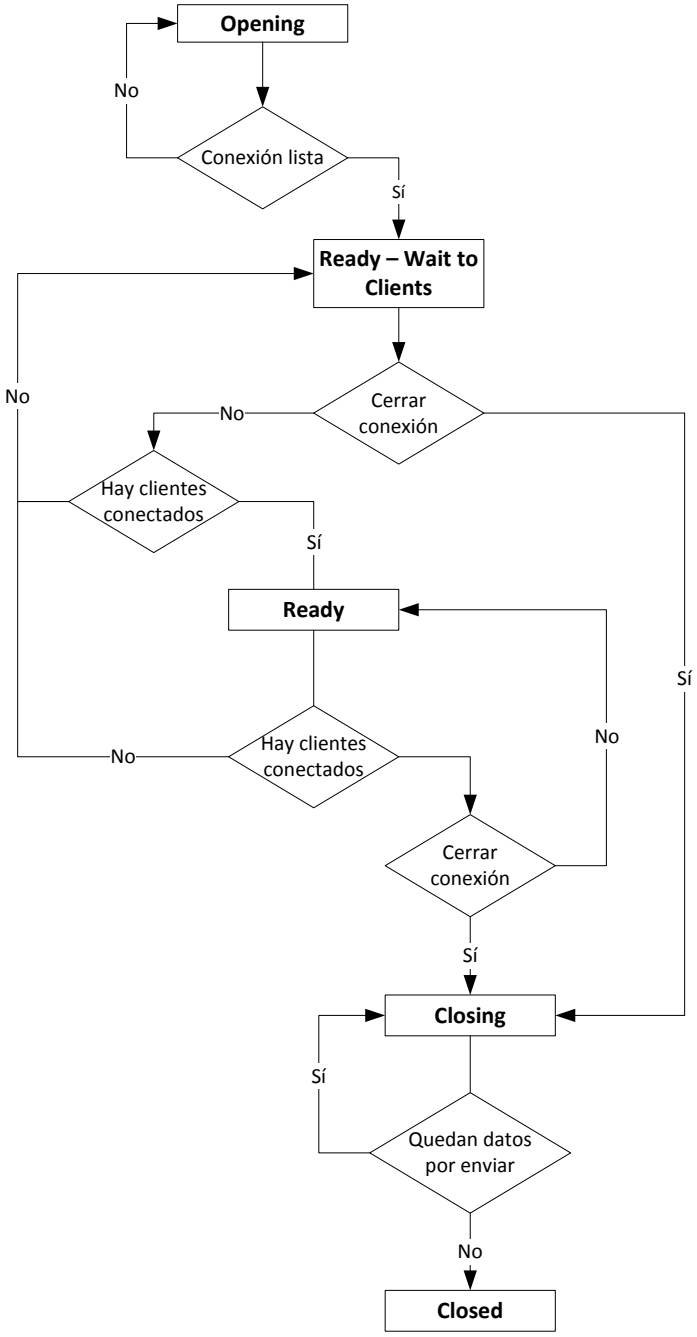

FIGURA 4.18: Evolución del estado de una conexión de tipo servidor

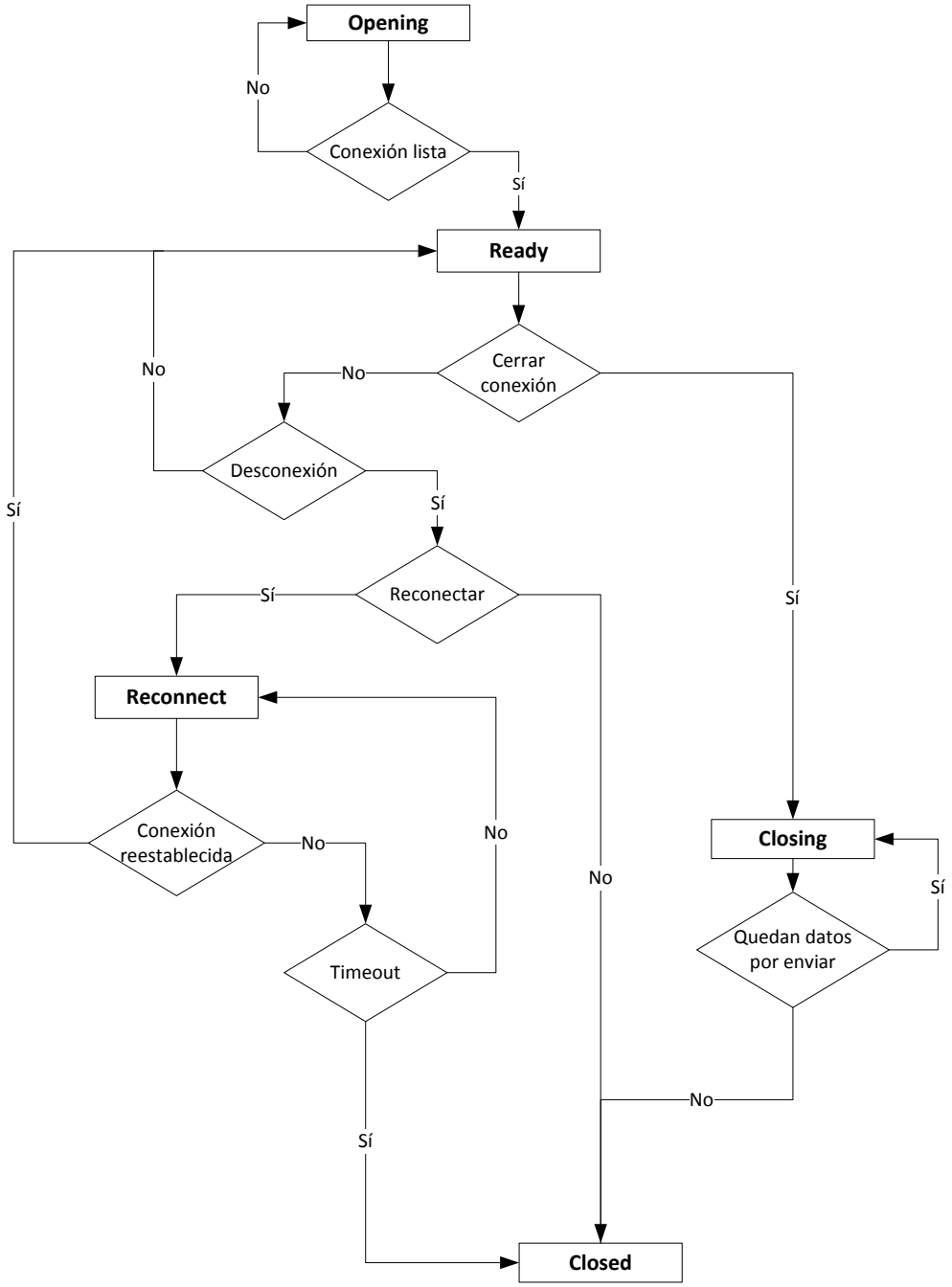

FIGURA 4.19: Evolución del estado de una conexión de tipo cliente

Ambos diagramas comparten los estados Opening, Ready, Closing y Closed. Su significado es el siguiente:

- en el estado Opening, la conexión está siendo establecida. Aún no se dispone de todos sus recursos.
- en el estado Ready la conexión está lista para recibir y transmitir datos. Por lo general, las conexiones permanecen en este estado durante la mayor parte del tiempo.
- en el estado Closing, el cliente ha solicitado el cierre de la conexión. Este estado es imprescindible para garantizar que se envían los datos pendientes. En cualquier caso, es importante

notar que se descartarán todos los paquetes recibidos y todos los nuevos paquetes que se desee enviar.

- en el estado Closed, la conexión está cerrada, y se han liberado todos sus recursos.

Las diferencias están relacionadas con el establecimiento de la conexión y las reconexiones.

Como ya hemos mencionado, una conexión de tipo servidor sólo está lista cuando uno o más clientes se conectan. En el estado Ready – Wait to clients, la conexión de tipo servidor está lista para aceptar conexiones entrantes, pero al no haber clientes conectados no será posible enviar ni recibir datos.

Por otra parte, las reconexiones sólo tienen sentido en conexiones de tipo cliente. Cuando la reconexión está habilitada, en el estado Reconnect se intentará restablecer la conexión con el servidor utilizando retroceso exponencial binario truncado. Si no se consigue tras quince intentos (suponen unos cinco minutos aproximadamente), se asumirá que la conexión está cerrada.

4.5.3.6. Hilos de red

Utilizando lo que acabamos de contar, podemos crear y manipular conexiones de red a un nivel de abstracción razonablemente elevado. No obstante, hasta ahora hemos omitido, para facilitar la comprensión del diseño, un aspecto fundamental: el rendimiento.

En la librería de red `twisted`, todos los paquetes entrantes se procesarán en el bucle reactor. Mientras tiene lugar el procesamiento del paquete, no se enviará ni se recibirá nada. Esto supone un problema, ya que en ocasiones los subsistemas de *CygnusCloud* tienen tiempos de respuesta muy elevados. Por ejemplo, una petición de arranque de una máquina virtual tardará mucho en procesarse por toda la entrada/salida que conlleva.

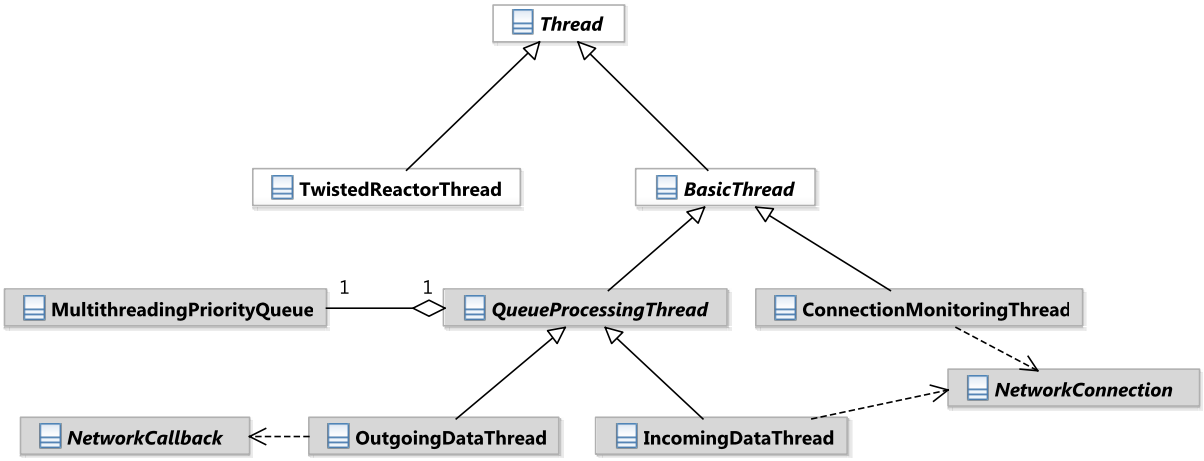

FIGURA 4.20: Jerarquía de hilos de red y sus relaciones más importantes

Así pues, resulta inadmisible que las comunicaciones se bloqueen durante el tiempo que tardan las peticiones en procesarse. La única forma de evitar esta situación es repartir la ejecución de las funciones de red entre varios hilos. Hemos realizado la siguiente descomposición:

- el bucle reactor de `twisted` se ejecutará en un hilo independiente. Esto permitirá a los usuarios utilizar el hilo principal de sus aplicaciones de la forma que estimen oportuna.

- cada conexión de red tendrá asociados un hilo de envío y un hilo de recepción de paquetes. El primero desencolará paquetes de la cola de envío y los transmitirá, y el segundo extraerá paquetes de la cola de recepción e invocará al código del cliente que los procesa.

- por motivos de eficiencia, el estado de *todas* las conexiones de red se actualizará en un hilo independiente.

El diagrama de clases de la figura 4.20 recoge las clases de la jerarquía de hilos de red y sus relaciones más relevantes.

La clase `BasicThread` añade a los hilos de Python la capacidad de detenerse cuando otros se lo solicitan, y la clase `QueueProcessingThread` añade también la capacidad de procesar elementos de una cola mediante un patrón *strategy*.

Por conveniencia, el hilo que ejecutará el bucle reactor, `TwistedReactorThread`, hereda directamente de `Thread`, la clase base de todos los hilos en Python. Por otra parte, la clase `ConnectionMonitoringThread` se corresponde con el hilo que actualizará el estado de todas las conexiones, y las clases `IncomingDataThread` y `OutgoingDataThread` se corresponden con los hilos de recepción y envío de paquetes.

Es importante notar que:

- es el hilo de recepción y no la conexión de red el que indica al cliente la recepción de un nuevo paquete. Para ello, se sigue utilizando la interfaz que define la clase `NetworkCallback`.
- todo el procesamiento de los paquetes entrantes tendrá lugar en un hilo `IncomingDataThread` y no en el hilo del bucle reactor de *twisted*, lo que nos permite garantizar que el tiempo de respuesta de la red será adecuado independientemente del tiempo que tarde en procesarse el paquete.

4.5.3.6.1. Control de la concurrencia

Si recapitulamos considerando todo lo que hemos visto, para poder comunicar varias máquinas entre sí es necesario disponer de los siguientes hilos:

- dos hilos, uno de envío y otro de recepción, para cada conexión.
- un hilo para actualizar el estado de las conexiones
- un hilo para ejecutar el bucle reactor

Además, para garantizar el correcto funcionamiento del sistema es necesario utilizar secciones críticas, para lo que necesitamos mecanismos de sincronización y, por tanto, aún más recursos.

Si el número de conexiones de red de cada máquina es reducido, no habrá problemas. Pero existe un tipo de máquina en CygnusCloud que estará conectada a muchas otras: el servidor de *cluster*.

En la sección 4.4.2.3 dijimos que una de sus funciones es realizar el balanceado de carga entre varios servidores de máquinas virtuales. A medida que el número de servidores de máquinas virtuales crece, los recursos asociados a los hilos y a los mecanismos de sincronización también lo harán, reduciendo considerablemente la escalabilidad. Por ejemplo, si el servidor de *cluster* está conectado a 10 servidores de máquinas virtuales, serán necesarios

$$10 \cdot 2 + 1 + 1 = 22 \text{ hilos de red}$$

junto con los mecanismos de sincronización correspondientes. Aunque la CPU de la máquina pueda lidiar con este número de hilos, muchos estarán compitiendo por entrar en las secciones críticas, lo que reduce el rendimiento. Por tanto, debemos reducir el número de hilos de la red.

Lo primero que debemos observar es que, con independencia del número de conexiones de red que haya, todo el tráfico viajará por el mismo medio físico. Por ello, si hacemos que todas las conexiones compartan el mismo hilo de envío (y, por tanto, la misma cola de envío) el rendimiento de la red no se resentirá significativamente.

Además, esto tiene una ventaja adicional: el tráfico prioritario, sea de la conexión que sea, siempre se enviará antes que el tráfico no prioritario. Si se usan varios hilos de envío, el tráfico de las distintas conexiones se mezclará en el bucle reactor, y no será posible garantizar este comportamiento.

Con este cambio, el número de hilos de la red se reduce en casi un 50 %, pero aún tenemos margen de mejora. En ocasiones, el código del cliente procesa los paquetes recibidos a través de varias conexiones de red de forma idéntica. Esto ocurriría si, por ejemplo, el servidor de *cluster* escribiese periódicamente el estado de todos los servidores de máquinas virtuales en un *log*.

En estos casos, podemos reducir el número de hilos y evitar muchos problemas de sincronización haciendo que las conexiones correspondientes compartan el hilo de recepción de paquetes (y, por tanto, también la cola de recepción de paquetes). Esta segunda mejora permite que, en el ejemplo anterior, sólo haya

$$2 + 1 + 1 = 4 \text{ hilos de red}$$

en el servidor de *cluster*, lo que incrementa la escalabilidad.

Para facilitar el uso de la red, lo más conveniente es detectar cuándo es posible realizar esta optimización y aplicarla de forma totalmente transparente para el usuario. Aunque detectar estos casos puede parecer complicado, en realidad no lo es: cuando el cliente utiliza el mismo objeto para procesar los paquetes recibidos a través de varias conexiones (es decir, el mismo *callback*), es posible aplicar la optimización de forma segura.

En cambio, si el cliente utiliza objetos diferentes o varias instancias de una misma clase, no es posible asegurar que estamos ante uno de estos casos, por lo que la optimización no se aplica.

Finalmente, puesto que los hilos y colas de recepción de paquetes se comparten, no podemos limitarnos a destruirlos cuando se cierra la conexión. Por ello, existe un contador de referencias para estos dos objetos, que sólo cuando este llega a cero.

4.5.3.7. La clase `NetworkManager`

A causa de las optimizaciones que hemos aplicado para reducir el número de hilos de la red, crear conexiones de red no resulta sencillo. Asimismo, tampoco resulta interesante mostrar la implementación de la red a los clientes, ni tampoco permitir que estos puedan crear paquetes libremente.

Para resolver estos problemas, hemos creado la clase `NetworkManager`. Sus objetivos son los siguientes:

- ocultar las optimizaciones que hemos realizado
- garantizar que todos los paquetes que envían los usuarios no comprometen el correcto funcionamiento de la red
- proporcionar una fachada que permite
  - establecer, utilizar y cerrar conexiones de red, y
  - enviar información a través de dichas conexiones

  a un elevado nivel de abstracción.

Las relaciones más relevantes de la clase `NetworkManager` aparecen en el diagrama de clases de la figura 4.21.

En el diagrama anterior no aparecen los hilos de recepción ni las colas de paquetes recibidos porque no están asociados a ningún objeto `NetworkManager`: estos objetos están siempre asociados a una conexión de red, de la que se tomarán en caso de que deban compartirse.

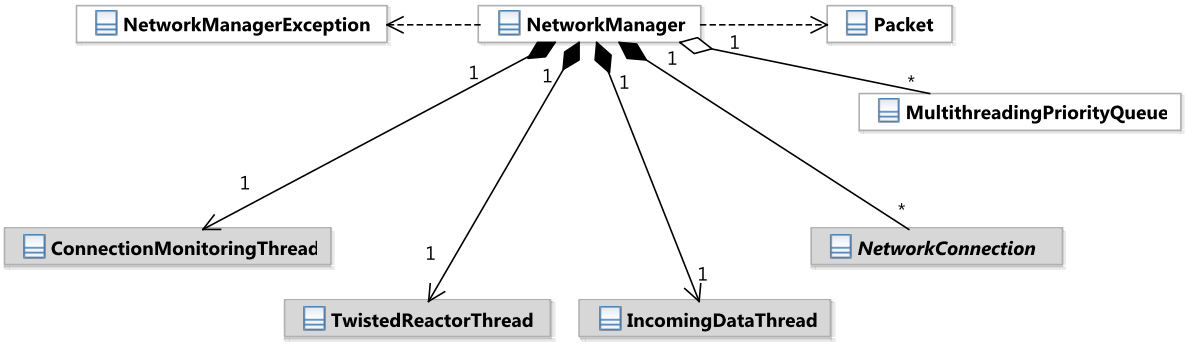

FIGURA 4.21: Relaciones más importantes de la clase `NetworkManager`

4.5.4. El paquete `ftp`

4.5.4.1. El servidor FTP `pyftpdlib`: visión general

Tal y como dijimos en la sección 4.4.14, `pyftpdlib` es un servidor FTP ligero y escrito íntegramente en *Python*. El diagrama de clases de la figura 4.22 muestra las clases que hemos utilizado para implementar nuestro servidor FTP y sus principales relaciones.

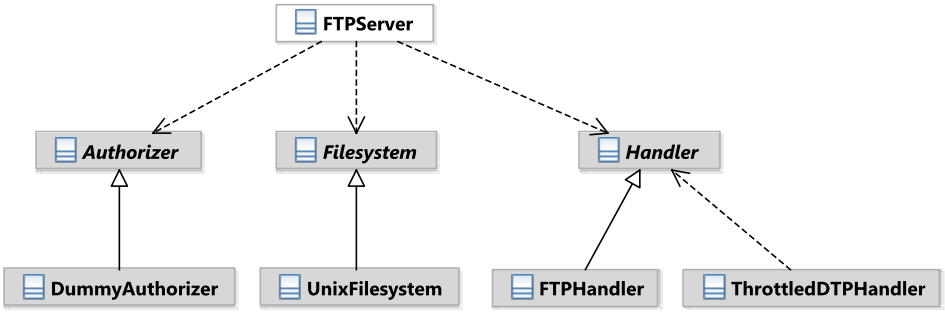

FIGURA 4.22: Principales clases del servidor FTP `pyftpdlib`

En primer lugar, la clase abstracta `Authorizer` define la interfaz que se usará para restringir el acceso al servidor FTP. Nuestro servidor FTP utiliza su subclase concreta `DummyAuthorizer`, que restringe el acceso al servidor FTP utilizando nombres de usuario y sus contraseñas que se transmiten sin cifrar.

Por otra parte, la clase abstracta `Filesystem` define la interfaz que se usará para manipular los permisos del sistema de ficheros. Nuestro servidor FTP utiliza su subclase concreta `UnixFilesystem`, ya que siempre se ejecutará en *Linux*, que es un sistema operativo tipo UNIX.

Asimismo, la clase abstracta `Handler` define la interfaz que se usará para procesar los eventos asociados a las transferencias de ficheros. Así,

- su subclase `FTPHandler` define la interfaz que se usará para generar los eventos asociados al servidor FTP (como, por ejemplo, las conexiones y desconexiones de usuarios). Creando una subclase de `FTPHandler`, podemos determinar la forma en que se procesan estos eventos.
- su subclase `ThrottledDTPHandler` permite controlar el ancho de banda consumido por el tráfico FTP.

Finalmente, la clase `FTPServer` implementa el resto del código del servidor FTP.

4.5.4.1.1. Configuración de un servidor FTP

Para configurar un servidor FTP basado en `pyftpdlib`, es necesario determinar

- la forma en que se hará la autenticación en el servidor FTP,
- la forma en que se procesarán los eventos asociados a las transferencias FTP y, opcionalmente,
- la porción del ancho de banda total que podrá ser utilizado por el tráfico FTP.

Por tanto, basta con instanciar cuatro clases: la clase `FTPServer`, una subclase concreta de `Authorizer`, una subclase de `FTPHandler` y, opcionalmente, la clase `ThrottledDTPHandler`.

4.5.4.1.2. Eventos FTP generados por el servidor FTP `pyftpdlib`

Como ya hemos mencionado, el servidor FTP `pyftpdlib` genera eventos para que el código cliente pueda responder adecuadamente cuando falla una transferencia FTP, cuando se conecta un nuevo cliente, etcétera. Los eventos FTP que se generan y su información asociada son los siguientes:

- desconexión inesperada de un cliente. No tiene asociado ningún tipo de información.
- conexión de un cliente. Tiene asociado el nombre de usuario que ha empleado el cliente para conectarse al servidor.
- desconexión normal de un cliente. Tiene asociado el nombre de usuario que ha empleado el cliente para conectarse al servidor.
- fin de envío de fichero. Tiene asociado el nombre del fichero que ha terminado de enviarse.
- fin de recepción de fichero. Tiene asociado el nombre del fichero que se ha recibido.
- error en el envío de un fichero. Tiene asociado el nombre del fichero que ha tratado de enviarse.
- error en la recepción de un fichero. Tiene asociado el nombre del fichero que se ha recibido parcialmente.

4.5.4.2. El servidor FTP de *CygnusCloud*

El servidor FTP de *CygnusCloud* utiliza como base el servidor FTP `pyftpdlib`. Sus principales características son las siguientes:

- permite realizar la gestión de usuarios y contraseñas en tiempo de ejecución. Como ya hemos dicho, estos datos siempre se transmitirán sin cifrar.
- permite restringir el número total de conexiones al servidor FTP, y también el número máximo de conexiones por cada dirección IP.
- permite restringir el tráfico FTP a cierta interfaz de red.
- permite configurar la fracción del ancho de banda que usará el tráfico FTP. Para ello, detecta el ancho de banda del enlace por el que viajará el tráfico FTP.
- hace posible que los eventos FTP se atiendan en el código cliente cuando sea oportuno.
- utiliza un hilo dedicado para el servidor FTP.

El diagrama de clases de la figura 4.23 muestra las clases del servidor FTP y sus relaciones con las clases del servidor FTP `pyftpdlib`. Por claridad, en él hemos omitido las relaciones existentes entre las clases de `pyftpdlib`. Asimismo, hemos abreviado `CygnusCloud` como `CC`.

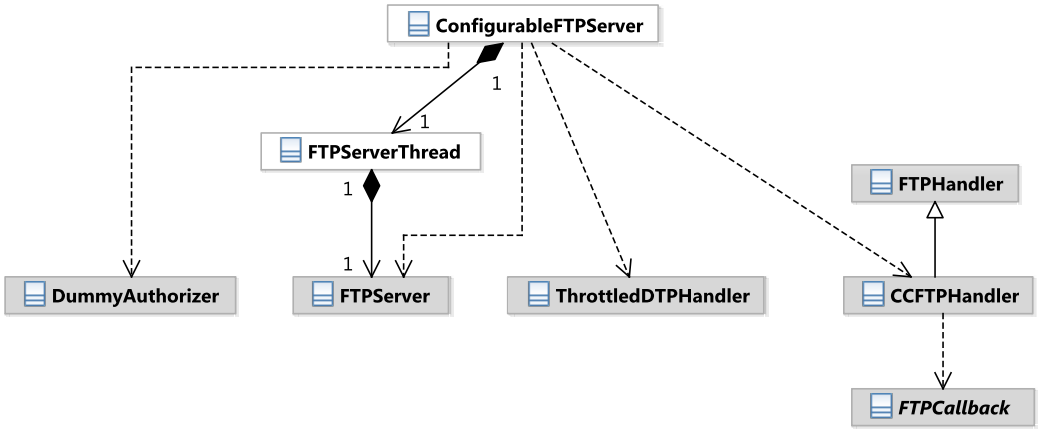

FIGURA 4.23: El servidor FTP de *CygnusCloud*: diagrama de clases

`ConfigurableFTPServer` es la clase principal del servidor FTP. Su misión es ocultar al código cliente el proceso de configuración del servidor FTP `pyftpdlib`, haciendo posible que este pueda procesar los eventos generados por el servidor FTP.

El procesamiento de dichos eventos se realiza a través de la interfaz definida por la clase abstracta `FTPCallback`. La clase `CygnusCloudFTPHandler`, que hereda de `FTPHandler`, se encargará de invocar a los métodos de un objeto `FTPCallback` a medida que se generen los eventos FTP.

Por otra parte, la clase `ConfigurableFTPServer` utiliza un objeto `DummyAuthorizer` y un objeto `ThrottledDTPHandler` durante el proceso de configuración del servidor FTP `pyftpdlib`, lo que significa que

- la autenticación usará nombres de usuario y contraseñas en texto plano, y
- el ancho de banda utilizado por el tráfico FTP podrá restringirse.

Finalmente, el código del servidor FTP se ejecuta en un hilo independiente, que se corresponde con la clase de hilo `FTPServerThread`. Así, el hilo en el que se haya configurado y arrancado el servidor FTP podrá utilizarse con otros fines.

4.5.4.3. El cliente FTP de *CygnusCloud*

El cliente FTP de *CygnusCloud* proporciona una interfaz que permite utilizar el cliente FTP de la librería estándar de *Python* a un mayor nivel de abstracción.

Su clase principal es `FTPClient`. A través de la interfaz de esta clase, es posible

- crear y destruir conexiones con un servidor FTP,
- subir un fichero a un determinado directorio del servidor FTP, y
- descargar un fichero ubicado en cierto directorio del servidor FTP.

Es importante notar que el cliente FTP de la librería estándar de *Python* *no* soporta FTP seguro. De todos modos, resulta muy sencillo utilizar otro cliente FTP que sí lo soporte: basta con modificar la implementación de los métodos de la clase `FTPClient`.

4.5.5. El paquete `imageRepository`

Tal y como dijimos en la sección 4.4.2.2, el repositorio de imágenes se ocupa de almacenar todas las imágenes que pueden utilizarse en un *cluster*, es decir, las imágenes que utilizan los alumnos, las imágenes que se están configurando y las imágenes base.

En esta sección, mostraremos detalladamente el diseño del demonio del repositorio de imágenes, que es el proceso que reside en el servidor que actúa como repositorio de imágenes y que atiende todas las peticiones recibidas por esa máquina.

Por conveniencia, y salvo que nos refiramos al servidor que actúa como repositorio de imágenes, a partir de ahora llamaremos directamente a este proceso repositorio de imágenes.

4.5.5.1. Adición de una capa adicional al servidor FTP

El repositorio de imágenes alberga un servidor FTP. Para intercambiar imágenes de disco con esta máquina, los servidores de máquinas virtuales se conectarán como clientes ese servidor FTP.

Ahora bien, si usamos directamente un servidor FTP para implementar el repositorio de imágenes, tendremos que enfrentarnos con serios problemas. Para ilustrarlos, daremos varios ejemplos.

En primer lugar supongamos que, en un momento dado,

- el *cluster* contiene n servidores de máquinas virtuales, que
- un administrador decide desplegar la misma imagen en todos ellos, y que
- ningún servidor de máquinas virtuales tiene almacenada esa imagen.

Cuando se realiza el despliegue de la imagen, el repositorio de imágenes tendrá que lidiar con n conexiones simultáneas, y debe transferir un fichero de gran tamaño por cada una de ellas. A medida que aumente el número de servidores de máquinas virtuales del *cluster*, tarde o temprano se acabarán generando errores por *timeout*.

Así, al crecer el número de servidores de máquinas virtuales del *cluster*,

- la infraestructura se hace más difícil de implementar o administrar, ya que hay que limitar el número de transferencias simultáneas entre el repositorio de imágenes y los servidores de máquinas virtuales, y
- se está desperdiciando cada vez más ancho de banda: el que consumen las transferencias de ficheros que fallan.

Para que sea cómodo utilizar el sistema *CygnusCloud*, si utilizamos directamente un servidor FTP resulta imprescindible limitar el número máximo de servidores de máquinas virtuales del *cluster*.

Así, el número máximo de servidores de máquinas virtuales del *cluster* pasará a estar limitado por el ancho de banda disponible para intercambiar imágenes con el repositorio de imágenes, y no por la capacidad del servidor de *cluster* para atenderlos a todos. Así, esto compromete la escalabilidad del sistema.

Por otra parte, el uso directo de un servidor FTP en el repositorio de imágenes tiene un inconveniente aún mayor: la aparición de conflictos en la edición de imágenes. Por ejemplo, supongamos que

- dos profesores A y B imparten la misma asignatura, que tiene asociada la configuración C_1, que
- tras utilizar C_1 con una máquina virtual, A y B descubren por separado que varias herramientas están mal configuradas, y que
- A y B deciden resolver por su cuenta el problema.

Para resolver este problema, tanto A como B tendrán que editar la imagen C_1. Pero como no se realiza ningún tipo de control a la hora de transferir los ficheros modificados al repositorio, los cambios que ha hecho A acabarán sobreescribiéndose con los que ha hecho B o viceversa. Por tanto, uno de los profesores ha estado perdiendo el tiempo: las correcciones que ha introducido se perderán.

En definitiva, el uso directo de un servidor FTP en el repositorio de imágenes presenta serios inconvenientes. Para evitarlo, el demonio del repositorio de imágenes no se limita a configurar y a arrancar el servidor FTP, y añade una capa adicional para resolver los dos problemas que acabamos de estudiar.

4.5.5.2. Funciones soportadas por el repositorio de imágenes

El repositorio de imágenes alberga un servidor FTP, al que añade las siguientes funciones:

- gestión de los identificadores de las imágenes. Todas las imágenes que pueden utilizarse en el *cluster* tienen asociado un identificador único, y el repositorio de imágenes se ocupa de la gestión de estos identificadores.
- transferencia de una imagen *en exclusividad* a una máquina remota. Tras realizar una de estas transferencias, la imagen correspondiente no podrá transferirse a ninguna otra máquina hasta que el repositorio reciba los datos modificados. Esto permite evitar la aparición de conflictos como el que estudiamos en la sección anterior.
- recopilación del estado del servidor.
- control del número de transferencias simultáneas. El mecanismo utilizado garantiza que todas las peticiones de transferencia recibirán un tratamiento equitativo.

Naturalmente, el repositorio de imágenes también soporta la transferencia, la recepción y el borrado de imágenes. Para realizar parte de las dos primeras operaciones, delega en el servidor FTP.

4.5.5.3. La conexión de control

El enlace que conecta el repositorio de imágenes a la red troncal de la UCM será *full duplex*. Para aprovechar su ancho de banda al máximo, debemos utilizar todo el ancho de banda de bajada y todo el ancho de banda de subida que podamos, por lo que debemos intentar, en la medida de lo posible, realizar varias transferencias de subida y de descarga de ficheros en paralelo.

Por otra parte, el ancho de banda del enlace está limitado. Si permitimos que los servidores de máquinas virtuales se conecten sin control al servidor FTP, podrán generarse errores por *timeout* que supondrán, tal y como dijimos en la sección 4.5.5.1, un desperdicio del ancho de banda del enlace.

Así pues, debemos restringir el número máximo de transferencias simultáneas entre el servidor FTP y las máquinas remotas, y también debemos realizar las transferencias de subida y de descarga en paralelo cuando sea posible. Para ello, las máquinas remotas no se conectarán directamente al servidor FTP, y dialogarán previamente con el demonio del repositorio de imágenes a través de una conexión de control adicional.

4.5.5.4. Los *slots* de transferencia

Como dijimos en la sección 4.5.5.3, para no desperdiciar el ancho de banda del enlace resulta imprescindible restringir el número de transferencias simultáneas entre el repositorio de imágenes y las máquinas remotas. Para ello, hemos utilizado un mecanismo basado en *slots*.

Cada *slot* estará asociado a una transferencia activa. Por ejemplo, si en un instante dado existen n transferencias activas en total, se estarán utilizando n *slots*. El número máximo de *slots* de transferencia se especifica en el fichero de configuración del demonio del repositorio de imágenes.

Así, cuando se recibe una nueva petición de transferencia,

- si hay un *slot* libre, se marcará como ocupado y se iniciará la transferencia, y
- si no hay ningún *slot* libre, la petición correspondiente se encolará.

Tras finalizar una transferencia, se desencolará e iniciará la siguiente petición. Esto nos permite limitar el número máximo de transferencias activas entre el repositorio de imágenes y el resto de máquinas del *cluster*.

4.5.5.5. Clases principales

El diagrama de clases de la figura 4.24 muestra las principales clases del repositorio de imágenes y sus relaciones con otras clases. Por claridad, en el diagrama de clases

- `ImageRepository` aparece abreviado como `IR`,
- las relaciones de la clase `FTPServerCallback` aparecen en azul,
- las relaciones de la clase `CommandsCallback` aparecen en rojo,
- hemos omitido las relaciones existentes entre las clases que no forman parte del paquete `imageRepository`, y
- no hemos indicado los paquetes a los que pertenecen las clases que no están definidas en el paquete `imageRepository`.

En los sucesivos diagramas UML relacionados con el repositorio de imágenes, seguiremos abreviando `ImageRepository` como `IR`.

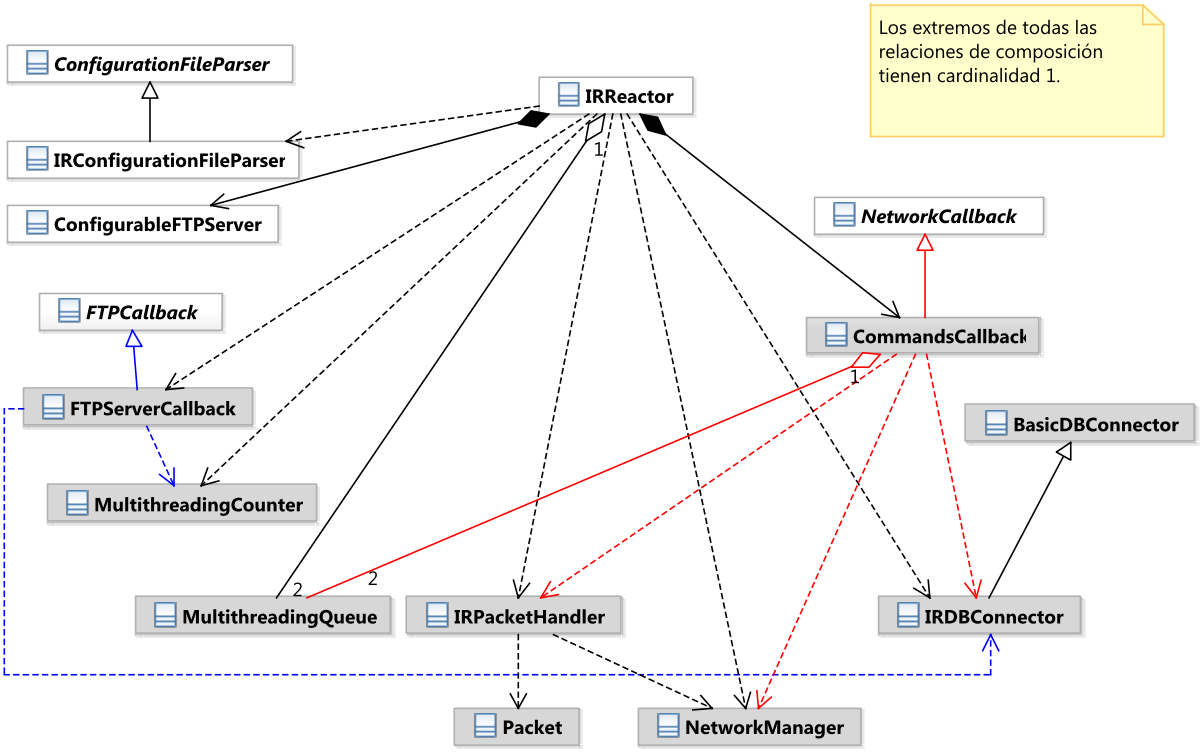

FIGURA 4.24: Principales clases del repositorio de imágenes y sus relaciones

Las principales clases que aparecen en el diagrama y sus responsabilidades son las siguientes:

- `ImageRepositoryReactor` es la clase principal del repositorio de imágenes. Además de ocuparse de los procesos de arranque y apagado del demonio del servidor de imágenes, también se encarga de asignación de *slots* a las transferencias encoladas.
- la clase `CommandsCallback` procesará todos los paquetes recibidos a través de la conexión de control. Análogamente, la clase `FTPServerCallback` procesará todos los eventos generados por el servidor FTP.
- la clase `ImageRepositoryPacketHandler` dispone de métodos para crear y leer los paquetes que se intercambian a través de la conexión de control.

- la clase `ImageRepositoryDBConnector` dispone de métodos para manipular la base de datos del repositorio de imágenes.
- la clase `ImageRepositoryConfigurationFileParser` procesa el fichero de configuración del demonio del repositorio de imágenes.

Por otra parte,

- las clases `ConfigurationFileParser`, `BasicDBConnector`, `MultithreadingQueue` y `MultithreadingCounter` forman parte del paquete `ccutils`,
- las clases `Packet`, `NetworkManager` y `NetworkCallback` forman parte del paquete `network`, y
- las clases `FTPCallback` y `ConfigurableFTPServer` forman parte del paquete `FTP`.

Más adelante, justificaremos el uso de estas clases, así como la cardinalidad de las relaciones de agregación y composición.

4.5.5.6. Arranque del repositorio de imágenes: secuencia básica

El proceso de arranque del demonio del repositorio de imágenes aparece en el diagrama de secuencia de la figura 4.25. En él, `ImageRepository` aparece abreviado como `IR`. Por claridad, estamos suponiendo que no se produce ningún error durante el proceso de arranque

Como puede observarse en el diagrama, desde el punto de entrada del demonio del repositorio de imágenes se invoca a varios métodos de la clase `ImageRepositoryReactor`. Los pasos que se siguen son los siguientes:

1. se crea la base de datos si no existe, y se registra un usuario que disponga de todos los permisos sobre ella.
2. se parsea el fichero de configuración. El reactor del repositorio de imágenes utilizará esa información durante el proceso de inicialización.
3. se crean las dos colas de transferencias y el contador de *slots*.

   Aunque sería posible utilizar una única cola de transferencias y extraer de ella peticiones de subida y de bajada, por eficiencia hemos utilizado dos colas: una contiene las peticiones de subida, y otra las peticiones de bajada. Esto explica la cardinalidad de las relaciones de agregación del diagrama de clases de la figura 4.24.

   Por otra parte, el contador de *slots* indica cuántos *slots* de transferencia están ocupados en un momento dado.
4. se establece la conexión con la base de datos del repositorio de imágenes.
5. se inicializan el gestor de red y el gestor de paquetes (una instancia de `ImageRepositoryPacketHandler`). Como dijimos en la sección 4.5.5.5, esta clase manipular los formatos de paquete asociados al repositorio de imágenes. Así, si es necesario modificar alguno de ellos basta con modificar el código de esta clase.
6. se configura e inicia el servidor FTP. Es importante notar que la contraseña se genera aleatoriamente en cada arranque.
7. se crea la conexión de control. Para ello, es necesario disponer de un objeto *callback* que procese los paquetes recibidos por la conexión de control. En el caso del repositorio de imágenes, se utiliza como objeto *callback* una instancia de la clase `CommandsCallback`.
8. el hilo principal empieza a asignar *slots* a las peticiones de transferencia encoladas.

Tras esto, el repositorio de imágenes está totalmente operativo.

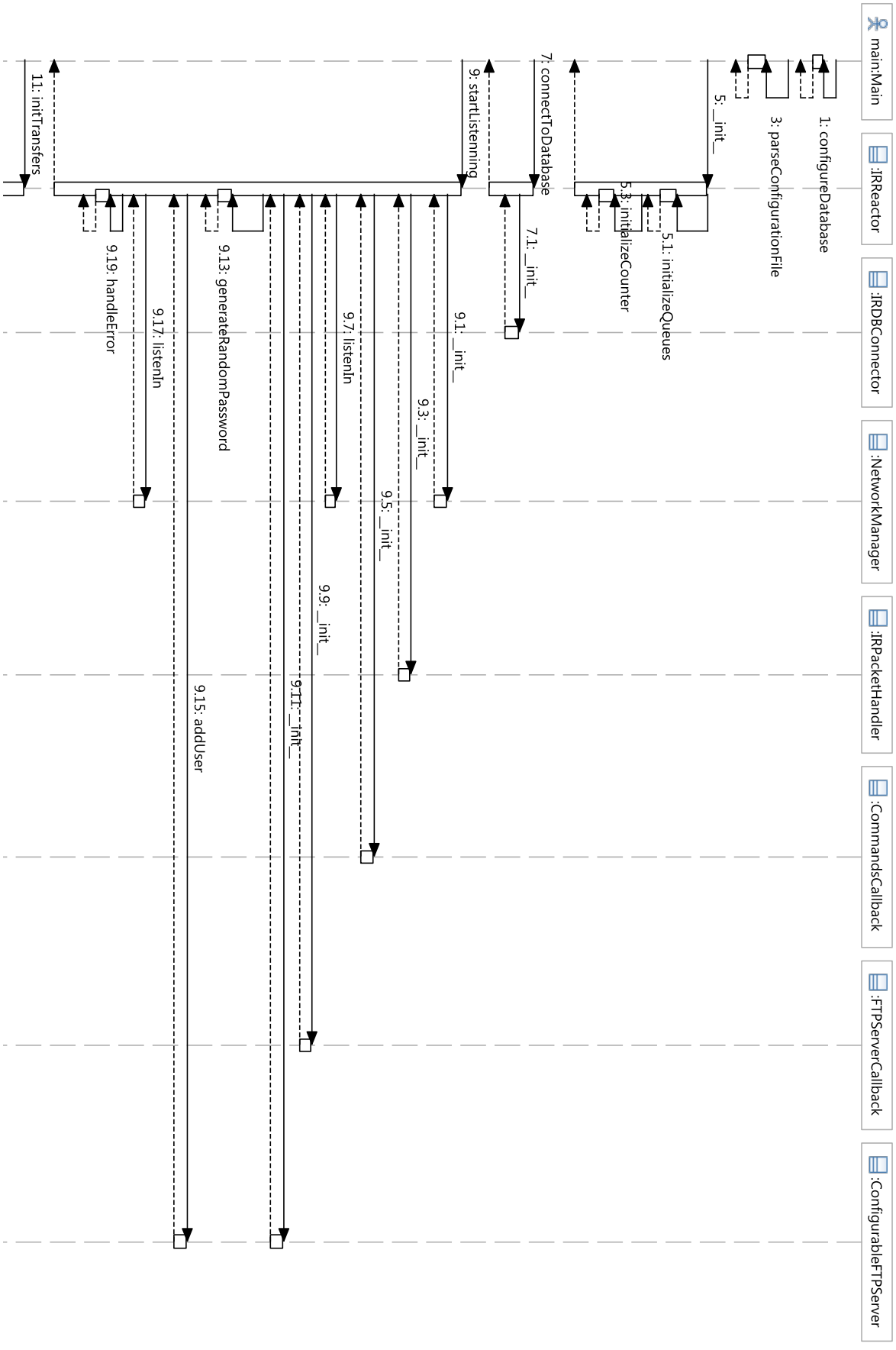

FIGURA 4.25: Arranque del repositorio de imágenes: secuencia básica

4.5.5.7. Arranque del repositorio de imágenes: tratamiento de errores

Mientras arranca el demonio del repositorio de imágenes, se producirán errores cuando

- el fichero de configuración del repositorio de imágenes sea incorrecto.
- la configuración de la base de datos puede falla.
- el puerto de la conexión de control o el puerto utilizado por el servidor FTP ya están en uso.
- la interfaz de red por la que circulará el tráfico FTP no está lista.

Todos estos errores se tratan de la siguiente manera:

1. tras detectar el error, se indica al hilo principal que se ha producido un error.
2. el hilo principal termina inmediatamente de iniciar transferencias.
3. el hilo principal para el servidor FTP y cierra la conexión de control.

Es importante notar que, en todos los procesos de apagado, el servicio de red nunca debe detenerse desde un hilo creado por la red. Si se intenta hacer esto, el hilo de red esperará hasta su propia terminación y, por tanto, se producirá un cuelgue.

4.5.5.8. Interacciones con una máquina remota

En esta sección, mostraremos todas las posibles interacciones que pueden tener lugar entre el repositorio de imágenes y una máquina remota que, dependiendo del caso, será un servidor de máquinas virtuales o el servidor de *cluster*.

Todas estas interacciones se basan en el intercambio de paquetes entre el repositorio de imágenes y la máquina remota a través de la conexión de control. Además, en aquellos casos en los que sea necesaria la transferencia de un fichero, también intervendrá el servidor FTP.

4.5.5.8.1. Registro de un identificador de imagen

Como dijimos en la sección 4.5.5.2, el repositorio de imágenes se ocupa de la gestión de los identificadores de las imágenes que pueden utilizarse en el *cluster*.

Por ello, cuando en un servidor de máquinas virtuales se cree una nueva imagen, será necesario solicitar el identificador de la misma al repositorio. El diagrama de secuencia de la figura 4.26 muestra cómo actúa el repositorio en estos casos.

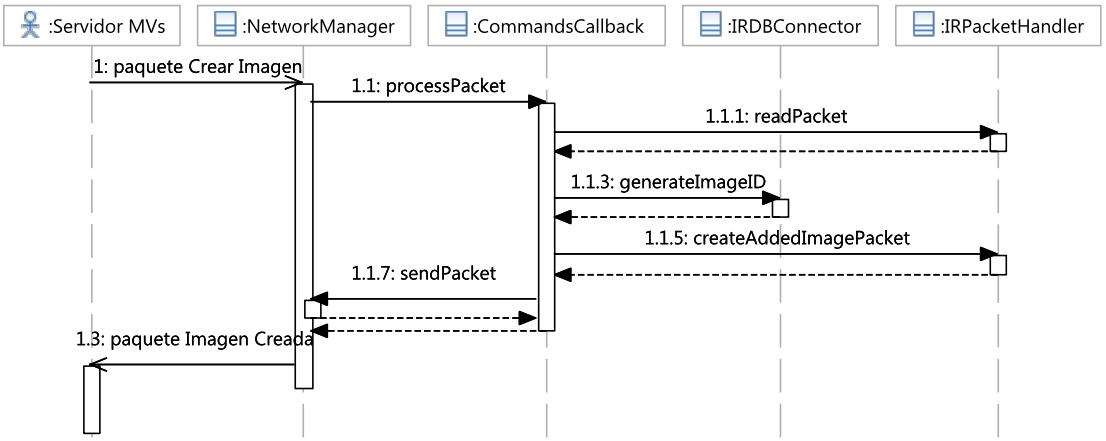

FIGURA 4.26: Registro de un identificador de imagen

Como puede observarse en dicho diagrama, se siguen los siguientes pasos:

1. el servidor de máquinas virtuales envía un paquete del tipo Crear Imagen al repositorio de imágenes. Estos paquetes no contienen ningún tipo de información adicional.
2. se lee el contenido del paquete recibido.
3. a partir del contenido de la base de datos, se genera un nuevo identificador de imagen.
4. se construye un paquete con la respuesta, que es del tipo Imagen Creada. Estos paquetes contienen el identificador de imagen generado.
5. se envía el paquete con la respuesta al servidor de máquinas virtuales.

Asimismo, en todas las interacciones con un servidor de máquinas virtuales, el repositorio de imágenes utiliza del modo de envío *unicast* de la conexión de control para enviar las respuestas. Por ello, sólo la máquina que envió la petición (y no todas las máquinas conectadas al repositorio de imágenes) recibirá la respuesta. Esto nos permite ahorrar mucho ancho de banda.

4.5.5.8.2. Descarga de imágenes

En la sección 4.4.15 dijimos que, para ahorrar ancho de banda y espacio en disco, las máquinas de la infraestructura no intercambian directamente ficheros de imagen, sino ficheros `.zip` que contienen ficheros de imagen.

Por ello, el repositorio de imágenes sólo intercambia ficheros `.zip` con los servidores de máquinas virtuales. En esta sección, mostraremos todas las interacciones relacionadas con la descarga de ficheros comprimidos desde el repositorio de imágenes.

Para empezar, nos centraremos en la interacción básica de inicio de una descarga, en la que no se producen errores. Esta interacción se recoge en el diagrama de secuencia de la figura 4.27.

Los pasos que se siguen son los siguientes:

1. el servidor de máquinas virtuales envía un paquete del tipo Petición `FTP RETR` al repositorio de imágenes. Este paquete contiene el identificador único de la imagen a descargar, y un *flag* que, en este caso, toma el valor `False`.
2. tras leer el contenido del paquete, el repositorio de imágenes comprueba que el fichero existe y que no ha sido concedido en exclusividad a otra máquina.
3. como no se producen errores, confirma a la máquina remota la recepción de su petición mediante un paquete del tipo Petición `FTP RETR` Recibida.
4. cuando un *slot* de transferencia queda libre y la petición llega a la cabecera de la cola,
   a) se comprueba nuevamente que el fichero puede transferirse. Si se detectase algún error, se abortaría inmediatamente la transferencia.
   b) se reserva un *slot* de transferencia
   c) se envía un paquete del tipo Inicio transferencia `FTP RETR` al servidor de máquinas virtuales. El paquete contiene toda la información que esta máquina necesita para conectarse al servidor FTP del repositorio de imágenes y descargar el fichero comprimido.

Para aprovechar al máximo el ancho de banda del enlace, las transferencias se desencolan procurando que, cuando sea posible, se realicen transferencias de subida y descarga en paralelo. Así, las transferencias no siempre se inician en estricto orden de recepción. Por claridad, hemos preferido no reflejar esto en el diagrama.

Por otra parte, no se podrá realizar la transferencia cuando

- el fichero a descargar no existe, o
- este ha sido concedido en exclusividad a otra máquina.

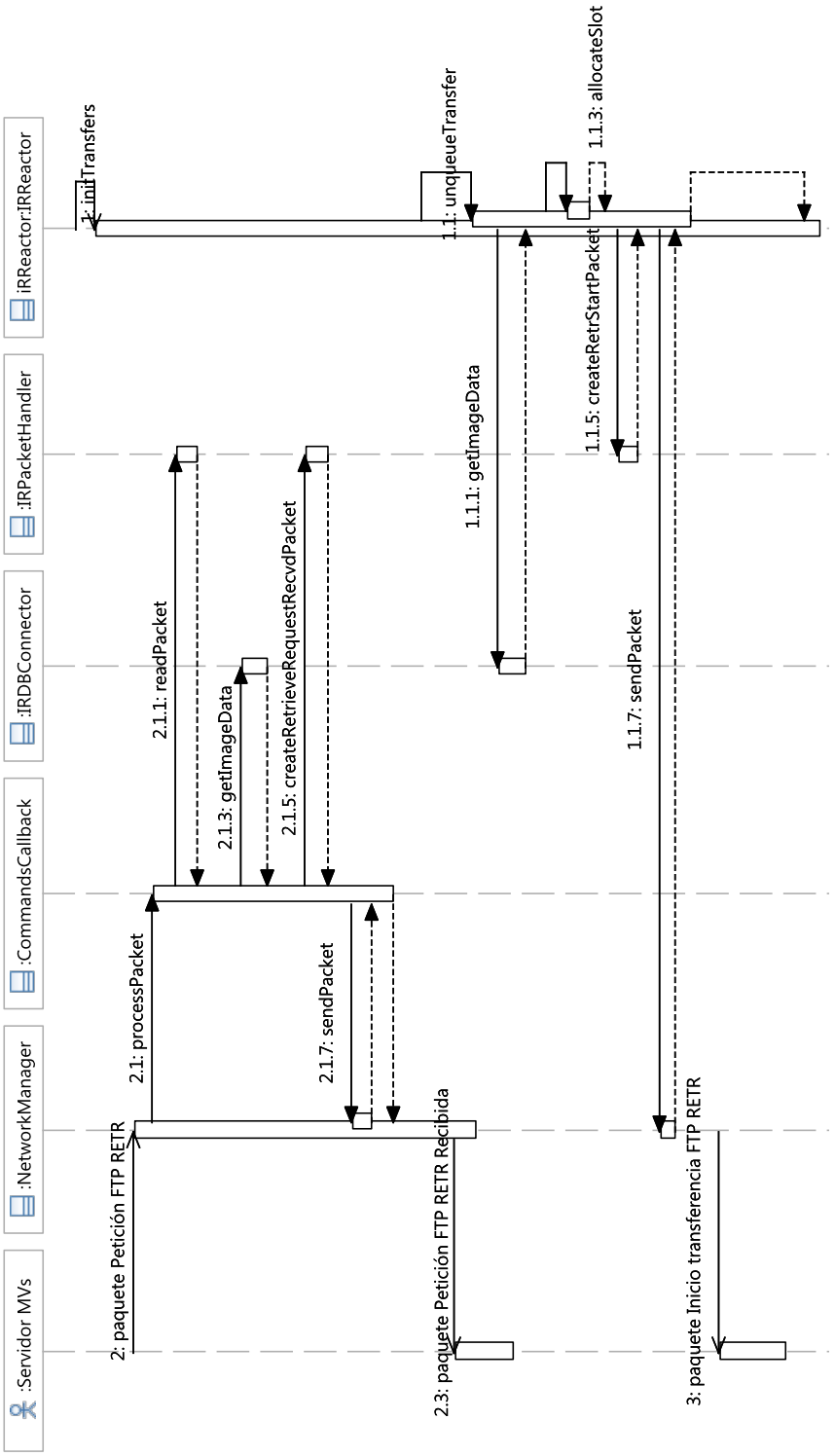

FIGURA 4.27: Inicio de la descarga de un fichero `.zip`

Estos errores pueden detectarse tanto al procesar el paquete Petición FTP RETR como al habilitar la transferencia. Los diagramas de secuencia de las figuras 4.28 y 4.29 muestran cómo interactúan el repositorio y el servidor de máquinas virtuales al detectar estos errores.

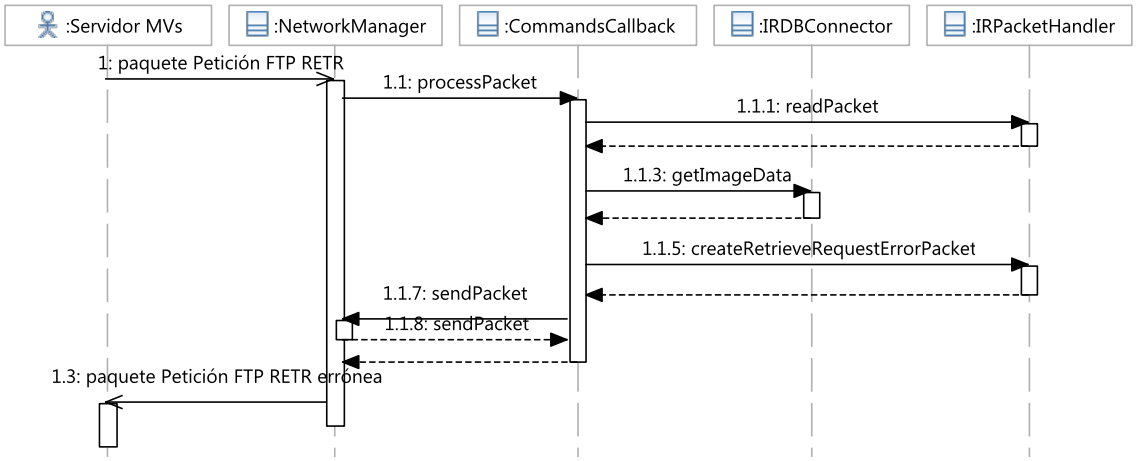

FIGURA 4.28: Inicio de la descarga de un fichero `.zip`: tratamiento de errores

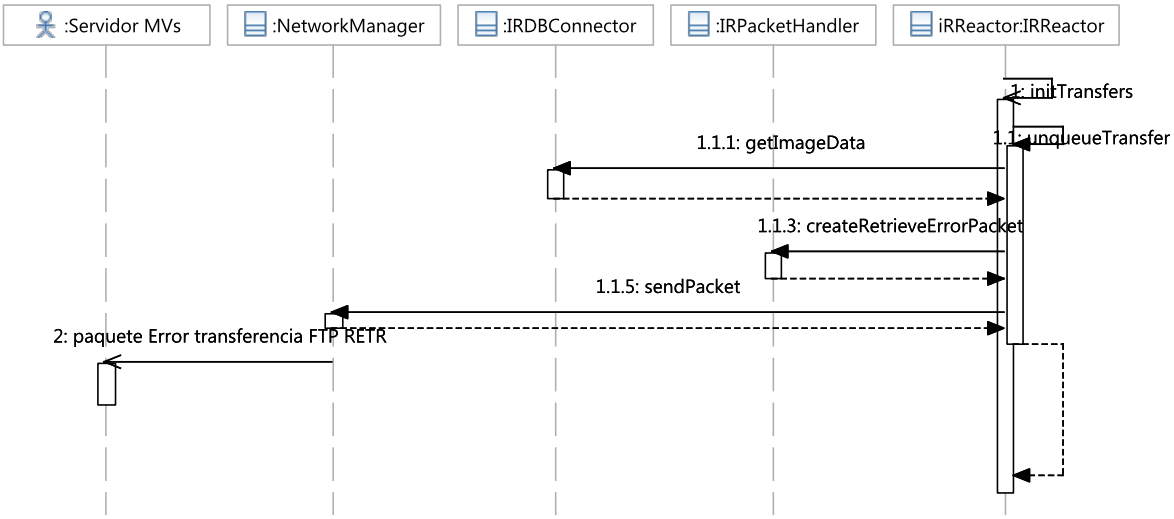

FIGURA 4.29: Detección de errores al habilitar la descarga de un fichero `.zip`

Siempre que se detecta un error

1. el repositorio de imágenes envía un paquete de error al servidor de máquinas virtuales. Estos paquetes serán de los tipos Solicitud FTP RETR errónea y Error en la transferencia FTP RETR, y contienen un código de descripción del error.
2. el servidor de máquinas virtuales aborta la transferencia.

Finalmente, cuando no se producen errores se realizará una transferencia FTP RETR entre el servidor de máquinas virtuales y el servidor FTP que reside en el repositorio de imágenes. Cuando esta finaliza, se liberará el *slot* que tiene asociado.

4.5.5.8.3. Uso de imágenes en exclusividad

Como dijimos en la sección 4.5.5.2, el repositorio de imágenes puede transferir en exclusividad un fichero comprimido a un servidor de máquinas virtuales. Esto significa que, mientras que ese

servidor de máquinas virtuales no lo libere, bien subiendo una versión modificada del mismo o bien pidiendo al repositorio que lo libere, ningún otro servidor de máquinas virtuales podrá descargarlo.

En esta sección, mostraremos cómo interactúa un servidor de máquinas virtuales para descargar un fichero desde el repositorio de imágenes en exclusividad. También estudiaremos cómo un servidor de máquinas virtuales puede liberar un fichero que ha descargado en exclusividad sin necesidad de transferirlo al repositorio de imágenes.

En primer lugar, estudiaremos la secuencia básica, en la que no se producen errores. El diagrama de secuencia de la figura 4.30 muestra la parte del proceso de descarga que cambia con respecto al caso anterior.

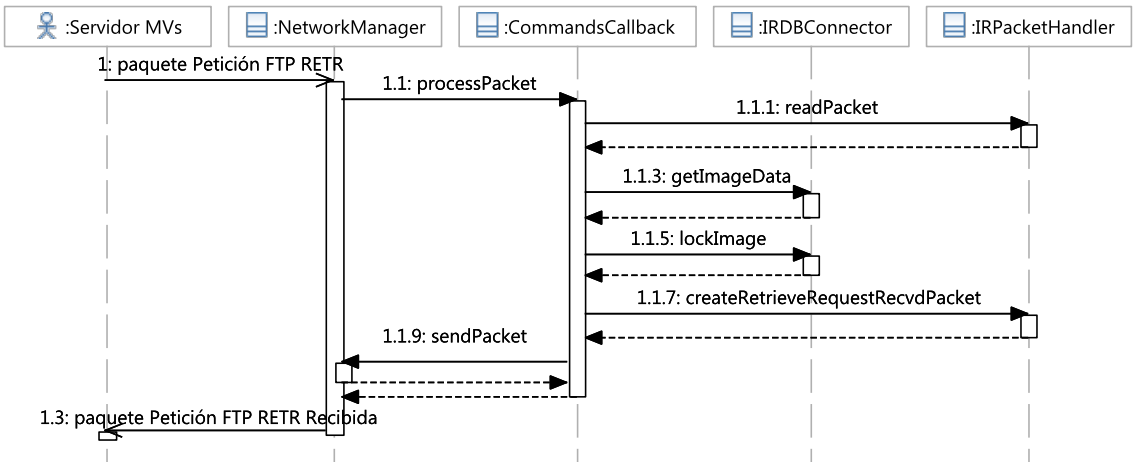

FIGURA 4.30: Descarga de un fichero `.zip` en exclusividad

Estas peticiones se procesan casi igual que una petición de descarga normal. No obstante, hay diferencias:

- el *flag* que forma parte todos los paquetes Petición `FTP RETR` toma el valor `True`.
- al procesar la petición, el repositorio de imágenes registra en la base de datos que la imagen se va a descargar en exclusividad.

Por otra parte, el diagrama de secuencia de la figura 4.31 muestra cómo un servidor de máquinas virtuales libera un fichero que tiene en exclusividad sin necesidad de transferirlo al repositorio.

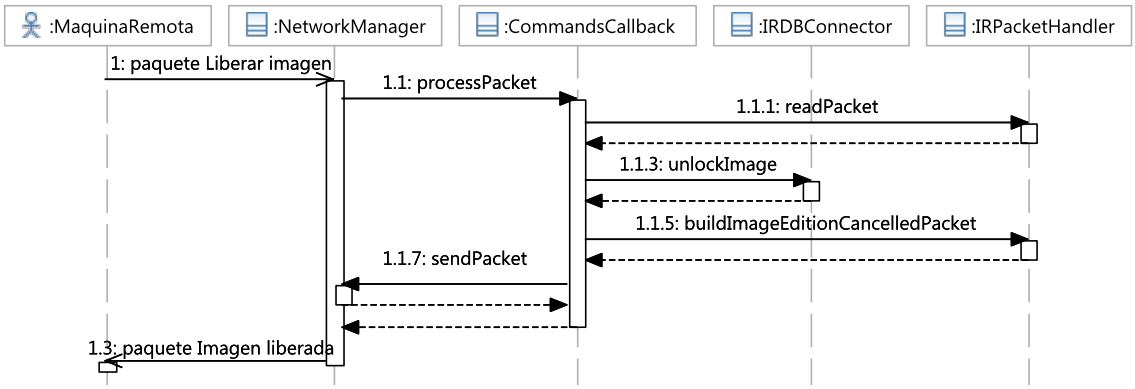

FIGURA 4.31: Liberación de un fichero descargado en exclusividad: secuencia sin transferencia

En este caso,

1. la máquina remota envía un paquete del tipo Liberar imagen al repositorio de imágenes. Este contiene el identificador del fichero comprimido.

2. al procesar el paquete, desde el objeto `CommandsCallback` se modifica el estado de la imagen en la base de datos del repositorio de imágenes.

3. finalmente, el objeto `CommandsCallback` envía un paquete del tipo Imagen liberada a la máquina remota.

Es importante notar que, cuando se intenta liberar un fichero que no está asignado en exclusividad a una máquina, la petición no tendrá efecto. Por ello, en este caso no se generarán errores.

4.5.5.8.4. Subida de imágenes

Las interacciones que tienen lugar para transferir una imagen de disco desde un servidor de máquinas virtuales al repositorio de imágenes son prácticamente idénticas a las interacciones de descarga. Por ello, en este apartado sólo comentaremos las diferencias.

El diagrama de secuencia de la figura 4.32 muestra la interacción básica de inicio de transferencia. En ella, no se produce ningún error. Como puede observarse en el diagrama, el repositorio de imágenes y la el servidor de máquinas virtuales intercambian los paquetes

- Petición `FTP STOR`, que contiene el identificador único de la imagen a transferir,
- Petición `FTP STOR` Recibida, e
- Inicio Transferencia `FTP STOR` que, nuevamente, contiene toda la información que necesita la máquina remota para conectarse al servidor FTP e iniciar la transferencia.

Asimismo, si un servidor de máquinas virtuales descargó en exclusividad la imagen que se ha terminado de subir al repositorio de imágenes, esta se liberará.

Por otra parte, en este caso sólo se producirán errores cuando el identificador de la imagen que se pretende subir no existe. Y estos errores pueden detectarse tanto al recibir la petición de transferencia como al habilitar la subida del fichero.

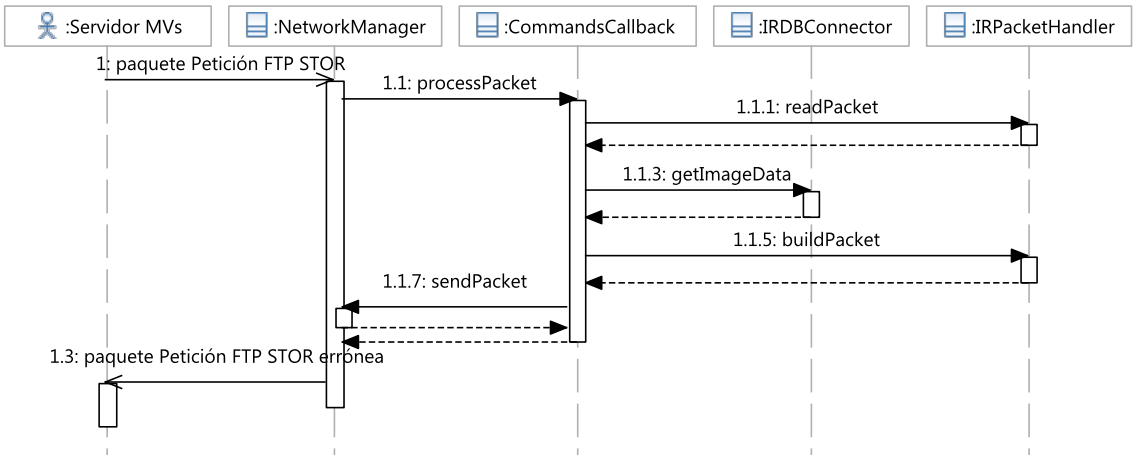

FIGURA 4.33: Detección de errores al recibir la petición de transferencia

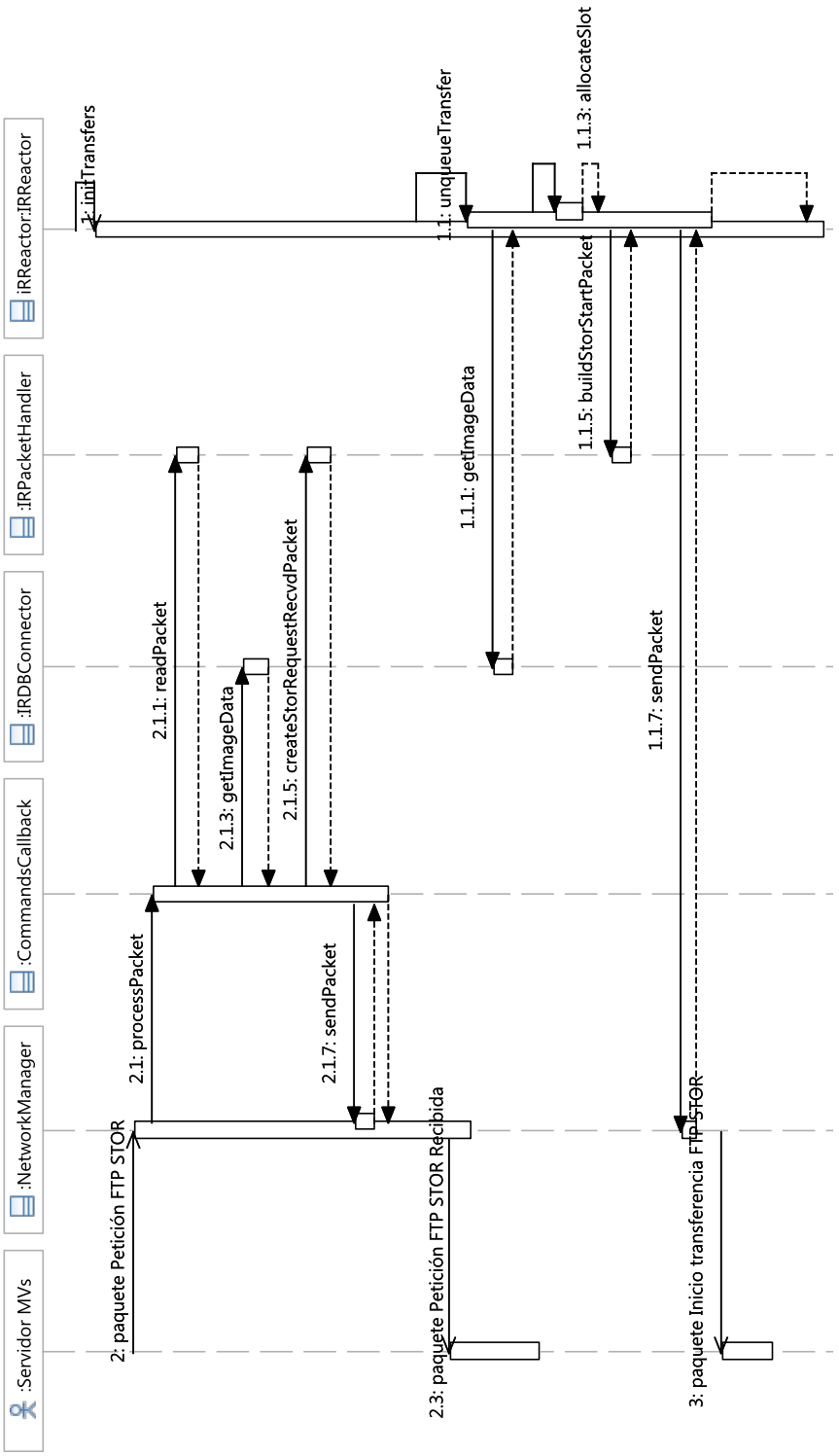

FIGURA 4.32: Inicio de la transferencia de una imagen desde una máquina remota: secuencia básica

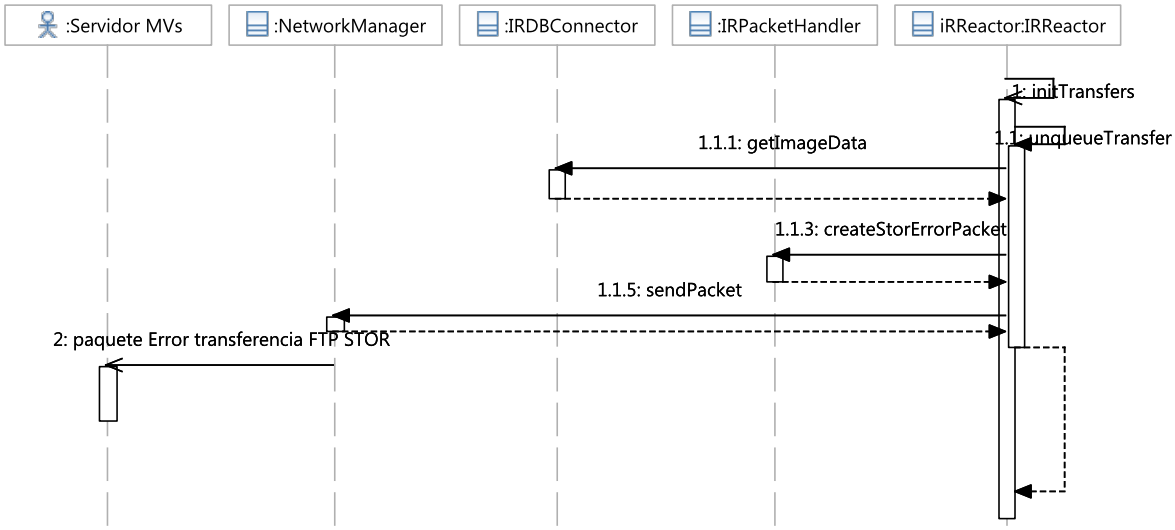

FIGURA 4.34: Detección de errores al habilitar la transferencia

Los diagramas de secuencia de las figuras 4.33 y 4.34 contienen las interacciones correspondientes. Como puede observarse en ambos, para informar del error se utilizan paquetes de los tipos Petición `FTP STOR` Errónea y Error en transferencia `FTP STOR`. Nuevamente, estos paquetes contienen el identificador asociado a las imágenes de disco y un código de descripción del error.

4.5.5.8.5. Transferencias parciales

Por claridad, en las dos secciones anteriores hemos omitido el tratamiento de un tipo de error adicional: los fallos en la transferencia FTP.

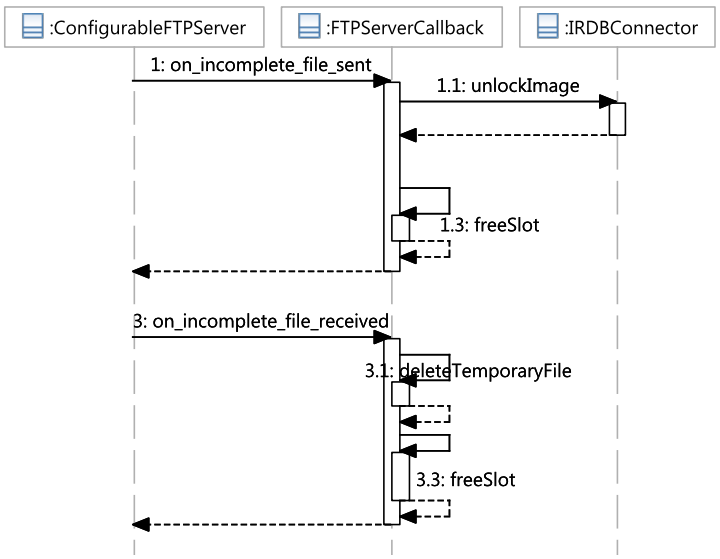

FIGURA 4.35: Tratamiento de las transferencias parciales

El diagrama de secuencia de la figura 4.35 muestra cómo se procesan los eventos de descarga parcial y de subida parcial en el repositorio de imágenes. Como se puede observar en él,

- cuando una transferencia de descarga falla, se libera el fichero correspondiente si estaba asignado en exclusividad a la máquina remota, y

- cuando una transferencia de subida falla, se borran los datos recibidos.

En ambos casos, se libera el *slot* de la transferencia. Además, no es necesario informar del error al servidor de máquinas virtuales, ya que el cliente FTP que este utiliza detectará el error.

4.5.5.8.6. Borrado de una imagen

Para borrar una imagen del repositorio, el servidor de *cluster* enviará a esta máquina un paquete del tipo Borrar imagen. El diagrama de secuencia de la figura 4.36 muestra todas las interacciones que pueden tener lugar al procesar estos paquetes.

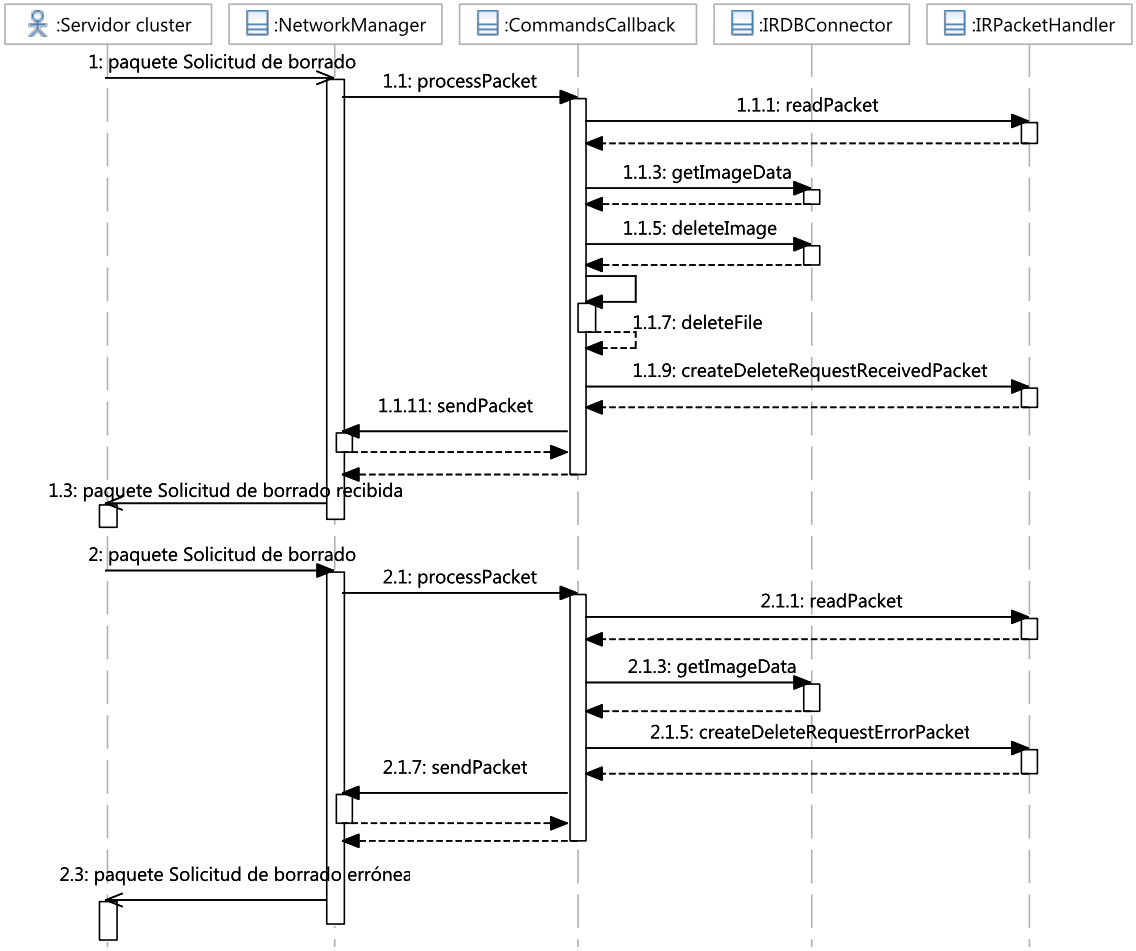

FIGURA 4.36: Borrado de una imagen del repositorio de imágenes

Los paquetes del tipo Solicitud de borrado contienen el identificador del fichero a borrar. Tras leer su contenido,

1. se comprueba que la imagen existe y que ningún servidor de máquinas virtuales la ha descargado en exclusividad.
2. si alguna de estas condiciones no se cumple, se generará y enviará un paquete del tipo Solicitud de borrado errónea al servidor de *cluster*. Este contiene el identificador de la imagen y un código de descripción del error.
3. si no se produce ningún error,
   a) se borra el fichero comprimido del disco

b) se borran los datos de la imagen de la base de datos, y

c) se crea y envía un paquete del tipo Solicitud de borrado recibida a la máquina remota. Este contiene el identificador único de la imagen borrada.

4.5.5.8.7. Solicitudes de estado

Para comprobar si puede atender las peticiones de creación y edición de imágenes, el servidor de *cluster* debe averiguar periódicamente el estado del repositorio de imágenes. Para ello, le envía un paquete del tipo Solicitud de estado.

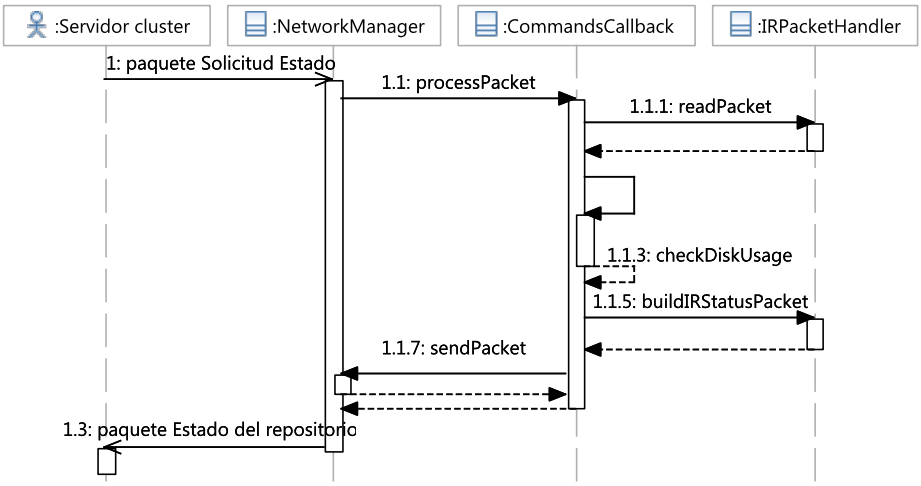

FIGURA 4.37: Recopilación del estado del repositorio

El diagrama de secuencia de la figura 4.37 muestra cómo interactúan el servidor de *cluster* y el repositorio de imágenes en estos casos. Como puede observarse en él, se siguen los siguientes pasos:

1. el servidor de *cluster* envía un paquete del tipo Solicitud de estado al repositorio de imágenes.

2. el demonio del repositorio de imágenes averigua el espacio en disco total y el espacio en disco disponible en el repositorio de imágenes.

3. con estos datos, se construye un paquete del tipo Estado del repositorio y se envía al servidor de *cluster*.

4.5.5.8.8. Apagado del repositorio de imágenes

El diagrama de secuencia de la figura 4.38 muestra el proceso de apagado del demonio del repositorio de imágenes. Como puede observarse en dicho diagrama, el apagado del repositorio es bastante sencillo:

1. cuando el servidor de cluster envía el paquete del tipo Apagado, el hilo principal deja inmediatamente de iniciar transferencias.

2. el hilo principal para el servidor FTP y detiene el servicio de red.

Nuevamente, y por las mismas razones que comentamos en la sección 4.5.5.7, el servicio de red deberá detenerse desde el hilo principal para evitar que se produzca un cuelgue.

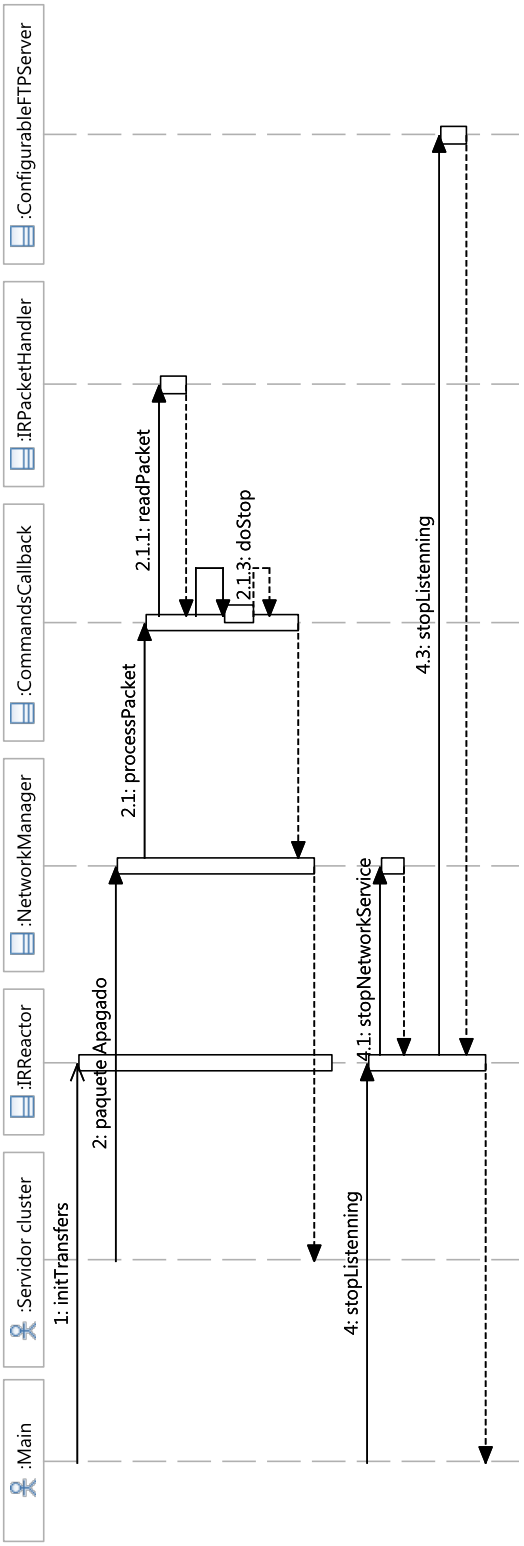

FIGURA 4.38: Apagado del repositorio de imágenes: diagrama de secuencia

4.5.5.9. Formatos de paquete

En esta sección, mostraremos detalladamente las características de los tipos de paquete asociados al repositorio de imágenes. El cuadro 4.5 muestra los códigos, prioridades y las constantes del tipo enumerado `PACKET_T` (definido en el paquete `imageRepository.packetHandling`) asociados a cada clase de paquete.

Como dijimos en la sección 4.5.3.3, cuanto menor es la valor numérico de la prioridad de un paquete, más prioritario es.

Tipo de paquete	Código	Prioridad	Constante (tipo enumerado)
Apagado	1	1	`HALT`
Crear imagen	2	5	`ADD_IMAGE`
Imagen creada	3	5	`ADDED_IMAGE_ID`
Solicitud `FTP RETR`	4	5	`RETR_REQUEST`
Solicitud `FTP RETR` recibida	5	5	`RETR_REQUEST_RECVD`
Solicitud `FTP RETR` errónea	6	4	`RETR_REQUEST_ERROR`
Inicio de transferencia `FTP RETR`	7	3	`RETR_START`
Error en la transferencia `FTP RETR`	8	4	`RETR_ERROR`
Solicitud `FTP STOR`	9	5	`STOR_REQUEST`
Solicitud `FTP STOR` recibida	10	5	`STOR_REQUEST_RECVD`
Solicitud `FTP STOR` errónea	11	4	`STOR_REQUEST_ERROR`
Inicio de transferencia `FTP STOR`	12	3	`STOR_START`
Error en la transferencia `FTP STOR`	13	4	`STOR_ERROR`
Solicitud de borrado	14	2	`DELETE_REQUEST`
Solicitud de borrado recibida	15	5	`DELETE_REQUEST_RECVD`
Solicitud de borrado errónea	16	4	`DELETE_REQUEST_ERROR`
Solicitud de estado	17	5	`STATUS_REQUEST`
Estado del repositorio	18	5	`STATUS_DATA`
Liberar imagen	19	2	`CANCEL_EDITION`
Imagen liberada	20	5	`IMAGE_EDITION_CANCELLED`

CUADRO 4.5: Características principales de los paquetes utilizados en el repositorio de imágenes

El contenido de los distintos tipos de paquete es el siguiente:

- los paquetes del tipo **Apagado** no contienen información adicional.
- los paquetes del tipo **Crear imagen** tampoco contienen información adicional.
- los paquetes del tipo **Imagen creada** contienen un nuevo identificador de imagen generado por el repositorio de imágenes.
- los paquetes del tipo **Solicitud** `FTP RETR` contienen el identificador único de la imagen a transferir, y un *flag* `modify` que indica si desea descargar el fichero en exclusividad o no.
- los paquetes del tipo **Solicitud** `FTP STOR` contienen el identificador único de la imagen a transferir.
- los paquetes de los tipos **Solicitud** `FTP RETR` **recibida** y **Solicitud** `FTP STOR` **recibida** contienen el identificador único de la imagen a descargar
- los paquetes de los tipos **Solicitud** `FTP RETR` **errónea**, **Solicitud** `FTP STOR` **errónea**, **Error en la transferencia** `FTP RETR` y **Error en la transferencia** `FTP STOR` contienen un código de descripción del error.
- los paquetes del tipo **Inicio de transferencia** `FTP RETR` e **Inicio de transferencia** `FTP STOR` contienen todo lo que necesita una máquina remota para subir o descargar una imagen desde el servidor FTP del repositorio de imágenes, es decir

    - el nombre de usuario y la contraseña del servidor FTP
    - el identificador único de la imagen a transferir
    - el puerto en el que escucha el servidor FTP,
    - el directorio del repositorio de imágenes donde se encuentra el fichero a transferir y
    - el nombre del fichero a transferir.

- los paquetes del tipo **Solicitud de borrado** contienen el identificador de la imagen a borrar.
- los paquetes del tipo **Solicitud de borrado recibida** contienen el identificador único de la imagen borrada.
- los paquetes del tipo **Solicitud de borrado errónea** contienen el identificador único de la imagen a borrar y un código de descripción del error.
- los paquetes del tipo **Solicitud de estado** no contienen información adicional
- los paquetes del tipo **Estado del repositorio** contienen el espacio en disco total y el espacio en disco utilizado en el repositorio de imágenes (en *kilobytes*).
- los paquetes del tipo **Liberar imagen** e **Imagen liberada** contienen el identificador único del fichero a liberar.

Finalmente, los *flags*, los identificadores de imágenes, los puertos y los códigos de error son valores enteros. El resto de datos que transportan los paquetes del repositorio de imágenes son *strings*.

4.5.5.10. Distribución de los ficheros

Cuando una transferencia FTP finaliza se llamará al método `on_file_sent()` o al método `on_file_received()` del objeto `FTPServerCallback`. Estos métodos reciben, como único argumento, la ruta del fichero cuya transferencia acaba de finalizar.

Como al finalizar ciertas transferencias hay que actualizar el estado de la imagen correspondiente en la base de datos del repositorio de imágenes es necesario que, a partir del nombre del fichero comprimido, sea posible obtener el identificador de la imagen.

Por ello, los nombres de los ficheros comprimidos no pueden fijarse arbitrariamente, y deben seguir cierto convenio. Por simplicidad, los nombres de todos los ficheros comprimidos que se almacenan en el repositorio de imágenes son de la forma

```
<identificador de las imágenes>.zip
```

Por ejemplo, el fichero comprimido asociado a la imagen `1` se llamará `1.zip`.

Por otra parte podemos aumentar la robustez del repositorio de imágenes si, cuando un servidor de máquinas virtuales se conecta al servidor FTP, sólo puede leer o escribir en un único directorio.

Por ello, en el directorio raíz del repositorio de imágenes existen múltiples subdirectorios. Cada uno de ellos contiene un único fichero comprimido, y sus nombres se fijan en base a las imágenes que contienen. Por ejemplo, la ruta del fichero comprimido `1.zip` será `<directorio raíz servidor FTP>/1/1.zip`.

4.5.5.11. Esquema de la base de datos

La base de datos del repositorio de imágenes contiene una única tabla, `Image`, cuyas columnas son las siguientes:

- `imageID`. Este valor entero es la clave primaria de la tabla, y es el identificador único de la imagen asociada al fichero comprimido.
- `compressedFilePath`. Se trata de un *string* de hasta 100 caracteres de longitud, que almacena la ruta absoluta del fichero comprimido con los datos de la imagen en el repositorio de imágenes.

- `imageStatus`. Este valor ocupa un *byte*, y codifica los tres posibles estados de las imágenes de disco, que aparecen en el cuadro 4.6.

  Para evitar errores, en el código fuente no hemos manipulado estos valores directamente, sino a través del tipo enumerado `IMAGE_STATUS_T`, definido en el paquete `imageRepository.database`.

Código	Constante (tipo enumerado)	Descripción
0	NOT_RECEIVED	El identificador de las imágenes de disco está reservado, pero no tiene asociado ningún fichero comprimido en el repositorio de imágenes.
1	READY	El repositorio de imágenes dispone del fichero comprimido. Además, ninguna máquina lo ha descargado en exclusividad.
2	EDITION	El repositorio de imágenes dispone del fichero comprimido. En este caso, una máquina lo ha descargado en exclusividad.

CUADRO 4.6: Codificación del estado de una imagen

4.5.6. El paquete `virtualMachineServer`

Como dijimos en la sección 4.4, los servidores de máquinas virtuales albergan

- las máquinas virtuales que utilizan los usuarios,
- las redes virtuales a través de las cuales las máquinas virtuales se conectan a la red troncal de la UCM, y
- las máquinas virtuales cuya configuración está siendo editada por un profesor o por un administrador.

En esta sección, presentaremos en detalle el diseño del demonio del servidor de máquinas virtuales. Este proceso, que residen en los servidores de máquinas virtuales, procesan todas las peticiones recibidas por estas máquinas.

Por conveniencia, y salvo que nos refiramos al servidor que actúa como servidor de máquinas virtuales, de ahora en adelante llamaremos a este proceso servidor de máquinas virtuales.

4.5.6.1. Redes virtuales

4.5.6.1.1. Creación de una red virtual en modo NAT

En todos los servidores de máquinas virtuales existe una red virtual configurada en modo NAT. A través de ella, las máquinas virtuales pueden acceder a la red troncal de la UCM.

Para crear las redes virtuales, el demonio del servidor de máquinas virtuales ejecuta durante su proceso de arranque las órdenes `virsh net-define` y `virsh net-create`.

La orden `virsh net-define` recibe como argumento la ruta de un fichero `.xml` en el que se define la configuración de la red virtual. La figura 4.39 muestra el contenido de un fichero de configuración asociado a una red virtual que opera en modo NAT.

```
<network>
    <name>default</name>
    <bridge name="virbr0" />
    <forward/>
    <ip address="192.168.122.1" netmask="255.255.255.0">
        <dhcp>
            <range start="192.168.122.2" end="192.168.122.254" />
        </dhcp>
    </ip>
</network>
```

FIGURA 4.39: Fichero de configuración de una red virtual operando en modo NAT

Como podemos observar, para configurar la red virtual basta con

- indicar el nombre de la red virtual.
- indicar el nombre de la interfaz virtual asociada al encaminador predeterminado. En el caso de la figura 4.39, esta se llamará `virbr0`.
- especificar la dirección IP y la máscara de red del encaminador predeterminado como atributos del elemento `ip`. Usando notación CIDR, en el fichero anterior se especifica que `192.168.122.1/24` será la dirección IP del encaminador predeterminado.
- indicar el rango de direcciones que asignará el servidor DHCP. Para ello, se utiliza el elemento `dhcp`.

Cuando se ejecuta la orden `virsh net-create`, se configuran convenientemente las redirecciones de datagramas en el *kernel* (a través de la orden `iptables`) y se lanza el proceso servidor DHCP (`dnsmasq`).

Tras la ejecución de la orden `virsh net-create`, toda máquina virtual que se conecte a la puerta de enlace podrá acceder a la red virtual y, por tanto, a la red troncal de la UCM.

Finalmente, las redes virtuales se destruyen ejecutando las órdenes `virsh net-destroy` y `virsh net-undefine`. La primera elimina las redirecciones de paquetes y mata el servidor DHCP, y la segunda borra la configuración de la red virtual.

Nótese que es posible destruir una red virtual manteniendo su configuración. En estos casos, es posible reactivar directamente la red virtual ejecutando la orden `virsh net-create`.

La clase `VirtualNetworkManager`

La clase `VirtualNetworkManager` define e implementa una interfaz que permite manipular redes virtuales a un elevado nivel de abstracción. El diagrama de clases de la figura 4.40 muestra sus dependencias.

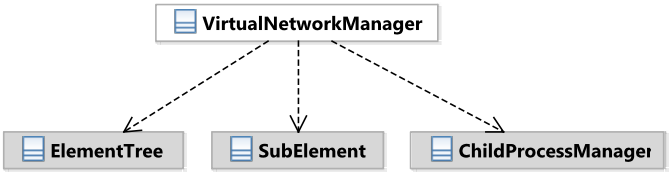

FIGURA 4.40: Relaciones de la clase `VirtualNetworkManager`

Las clases `ElementTree` y `SubElement` forman parte de la librería estándar de *Python*, y se utilizan para generar el fichero `.xml` con la configuración de la red.

Por otra parte, la clase `ChildProcessManager` dispone de métodos que permiten ejecutar comandos en *foreground* y *background*. La clase `VirtualNetworkManager` los utiliza para ejecutar los comandos `virsh net-define`, `virsh net-create`, `virsh net-undefine` y `virsh net-destroy`.

4.5.6.2. Interacción con `libvirt` a bajo nivel

En esta sección, mostraremos cómo se interactúa a bajo nivel con la librería `libvirt` para crear y destruir máquinas virtuales.

Para comenzar, explicaremos qué recursos son necesarios para arrancar una máquina virtual. Posteriormente, presentaremos el formato de los ficheros de configuración de máquinas virtuales y, finalmente, mostraremos la forma en que se interactúa con `libvirt` desde *Python*.

4.5.6.2.1. Recursos asignados a una máquina virtual

Toda máquina virtual debe tener asociados, como mínimo,

- una o más CPUs virtuales, todas pertenecientes a la misma arquitectura,
- una memoria principal, y
- dos imágenes de disco, en las que se almacenarán el *software* instalado y los datos de la máquina virtual respectivamente. En la sección 4.4.17 justificamos por qué es necesario utilizar dos imágenes de disco en lugar de una sola.

Aunque estos recursos son suficientes para que la máquina virtual arranque, no permiten interactuar con ella. Para que esta interacción sea posible, también es necesario

- utilizar un servidor VNC, que se usará para interactuar con la máquina virtual, o
- utilizar una conexión de red que permita interactuar con la máquina vía SSH (*Secure Shell*) o `telnet`.

Puesto que en la mayoría de casos los usuarios utilizarán interfaces gráficas de usuario para interactuar con la máquina virtual, el uso de un servidor VNC es imprescindible. Todo servidor VNC debe tener asignados los siguientes recursos:

- un puerto, que debe ser único dentro de cada servidor de máquinas virtuales. Este permitirá utilizar la máquina virtual mediante un cliente VNC convencional.
- un *websocket*, que también tiene asociado un puerto único dentro de cada servidor de máquinas virtuales. Los *websockets* permiten utilizar las máquinas virtuales utilizando el cliente VNC noVNC, del que hablamos en la sección 4.4.9.
- una dirección IP, que coincide con la dirección IP de una de las interfaces de red del servidor de máquinas virtuales.
- una contraseña, que impide que los usuarios ajenos a la máquina virtual puedan visualizar su pantalla o utilizarla.

Como dijimos en la sección 4.4.21, dado que la red troncal de la UCM es segura, por motivos de eficiencia el tráfico VNC no viajará cifrado. En caso de hacerlo, cada servidor VNC también tendría asociado un certificado SSL.

Además, los usuarios también necesitarán acceder a la red troncal de la UCM o a internet para poder introducir o extraer datos de la máquina virtual, por lo que es fundamental que esta esté conectada a una red virtual.

Para ello, la máquina virtual debe tener asignada una dirección MAC, que debe ser única dentro de la red virtual y, por ende, dentro del servidor de máquinas virtuales en el que reside.

Finalmente, para que `libvirt` pueda identificar correctamente todas las máquinas virtuales, estas deben disponer de

- un nombre, que debe ser único dentro del servidor de máquinas virtuales en el que reside la máquina, y
- un UUID (*Universally Unique Identifier*), que debe identificar a la máquina virtual de forma única en toda la infraestructura.

4.5.6.2.2. Formato de los ficheros de definición

Para crear una máquina virtual a través de libvirt es necesario reservar previamente todos los recursos que tiene asignados e incluirlos en un fichero .xml, llamado fichero de definición. En esta sección, explicaremos en el formato de estos ficheros.

```
<domain type='kvm'>
    <name>Squeeze_AMD64</name>
    <uuid>34ed2109-cd6d-6048-d47c-55bea73e39fd</uuid>
    <memory>1048576</memory>
    <currentMemory>1048576</currentMemory>
    <vcpu>1</vcpu>
    <os>
        <type arch='x86_64' machine='pc-1.0'>hvm</type>
        <boot dev='hd'/>
    </os>
    <features>
        <acpi/> <apic/> <pae/>
    </features>
    <clock offset='utc'/>
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>restart</on_reboot>
    <on_crash>restart</on_crash>
    <devices>
        <emulator>/usr/bin/kvm</emulator>
        <disk type='file' device='disk'>
            <driver name='qemu' type='qcow2' cache='writeback' io='native'/>
            <source file='/home/luis/SqueezeAMD64.qcow2'/>
            <target dev='vda' bus='virtio'/>
        </disk>
        <interface type='network'>
        <source network='default'/> <mac address='52:54:00:8a:56:41'/>
        </interface>
        <input type='tablet' bus='usb'/>
        <input type='mouse' bus='ps2'/>
        <graphics type='vnc' port='15010' websocket='15011' autoport='no' pass-
wd='CCRules!'
                keymap='es'>
            <listen type='address' address='192.168.0.5'/>
        </graphics>
        <video>
            <model type='cirrus' vram='9216' heads='1'/>
        </video>
    </devices>
</domain>
```

FIGURA 4.41: Fichero de definición de una máquina *Linux*

La figura 4.41 muestra el contenido de un fichero de definición utilizado para crear máquinas virtuales *Linux*. Las características más relevantes de este fichero son las siguientes:

- el atributo `type` del elemento `domain` indica la solución de virtualización que se utilizará para gestionar la máquina virtual. En nuestro caso toma el valor `kvm`, ya que utilizamos el hipervisor KVM.

  Es importante notar que las funciones que se pueden especificar en los ficheros de definición varían en función del sistema de virtualización que estemos usando.

- los elementos `name`, `uuid`, `vcpu` y `memory` especifican el nombre, el UUID, el número de CPUs virtuales y el tamaño de la memoria de la máquina virtual.

- esencialmente, el elemento `os` define la arquitectura del sistema operativo de la máquina virtual y la secuencia de arranque de la misma.

  En el caso del fichero de definición anterior, el sistema operativo podrá ser cualquier sistema operativo de 32 o 64 bits soportado por la arquitectura `x86_64`, y la máquina virtual arrancará directamente del primer disco duro.

- el elemento `features` permite activar las características ACPI (*Advanced Configuration and Power Interface*), APIC (*Advance Programmable Interrupt Controller*) y PAE (*Physical Address Extension*) de la máquina virtual.

  Sin entrar demasiado en detalles, podemos decir que la práctica totalidad de las CPUs `x86` con hasta diez años de antigüedad y todos los sistemas operativos existentes en el mercado soportan estas características, por lo que es mejor dejarlas activadas. En caso de que detecte alguna incompatibilidad, el sistema operativo de la máquina virtual podrá deshabilitarlas.

- el elemento `clock` permite configurar el desfase del reloj. Puede tomar dos valores: `localtime` y `utc`, que difieren en la forma en que se almacena la hora en la máquina virtual.

  Sin entrar en detalles, basta recordar que el elemento `clock` debe tomar el valor `localtime` en sistemas *Windows*, y `utc` en sistemas tipo UNIX.

- los elementos `on_poweroff`, `on_reboot` y `on_crash` determinan cómo actuar cuando la máquina virtual se apaga, se reinicia o falla respectivamente. En el fichero de definición de la figura 4.41, la máquina virtual se destruirá al apagarse, y se reiniciará en el resto de casos.

- el elemento `devices` permite definir la configuración del subsistema de entrada/salida de la máquina virtual. Sus principales elementos son los siguientes:

  - `emulator`, que especifica la ruta del fichero binario que emulará los dispositivos de entrada/salida de la máquina virtual. El emulador de KVM está ubicado en `/usr/bin/kvm`.
  - `disk`, que especifica la configuración de un disco duro. La máquina virtual asociada al fichero de definición de la figura 4.41 tiene un único disco duro, cuyos datos se extraen de una imagen de disco `qcow2`.

    Para obtener el mejor rendimiento posible, este disco duro se conecta al bus VirtIO que, como sabemos, proporciona un rendimiento muy similar al de un disco duro real.
  - `network`, que especifica la configuración de una tarjeta de red. La máquina virtual asociada al fichero de definición de la figura 4.41 se conectará a la red virtual `default`, utilizando para ello la dirección MAC `52:54:00:8a:56:41`.
  - `input`, que define un dispositivo de entrada. Habitualmente, los ficheros de definición especifican dos: un ratón PS/2 y una tableta gráfica USB que permiten el correcto movimiento del cursor cuando se utiliza el protocolo VNC.
  - `graphics`, que especifica la configuración del servidor VNC asociado a la máquina virtual. Entre otras cosas, es necesario especificar el puerto (`port`), el puerto del *websocket* nativo[2] (`websocket`), la contraseña (`passwd`), la distribución del teclado (`keymap`) y la dirección IP (atributo `address` del elemento `listen`) en la que escuchará el servidor VNC.

[2] Si no utilizamos los *websockets* nativos de QEMU, no es necesario incluir este atributo

Tal y como dijimos en la sección 4.4.8, estamos utilizando el servidor VNC integrado en KVM. Si utilizásemos otro, el elemento `graphics` no sería necesario, por lo que no aparecería en el fichero de definición.

- `video`, que define la configuración de la tarjeta gráfica de la máquina virtual. En el fichero de configuración de la figura 4.41, se utiliza una tarjeta gráfica `CLGD 5446` fabricada por *Cirrus Logic Inc.*, con 9 MB de memoria de vídeo y un único monitor conectado.

Por otra parte, los ficheros de definición de máquinas virtuales *Windows* son muy similares. Tan sólo es necesario dar el valor `localtime` al elemento `clock`: el resto de elementos se configuran como ya hemos visto.

Finalmente, es importante notar que el número de CPUs, el tamaño de la memoria RAM y el tamaño de las imágenes de disco vienen fijados por la familia de máquinas virtuales a la que pertenece la máquina virtual que se va a arrancar.

Estos datos se leerán de otro fichero de definición, que se usará como plantilla para crear todas las máquinas virtuales de la misma familia y que también se encuentra almacenado en el servidor de máquinas virtuales.

4.5.6.2.3. La clase `LibvirtConnector`

Para interactuar con `libvirt` desde *Python*, se utilizan dos tipos de llamadas a métodos:

- **llamadas síncronas**. En general, estas llamadas se utilizan para realizar consultas relacionadas con las máquinas virtuales activas (como, por ejemplo, para obtener los recursos asociados a cada una de ellas) y para enviar consultas al hipervisor (como, por ejemplo, las que leen el tiempo que cada máquina virtual ha usado la CPU).
- **llamadas asíncronas**. Sus resultados no están disponibles al finalizar la llamada, y se obtienen procesando los eventos asociados. Las llamadas síncronas se utilizan para realizar operaciones con una gran latencia, tales como el arranque y la destrucción de una máquina virtual.

Para paralelizar la implementación del servidor de máquinas virtuales es necesario definir una interfaz que permita interactuar con `libvirt` a un elevado nivel de abstracción. La clase `LibvirtConnector` define e implementa dicha interfaz. El diagrama de clases de la figura 4.42 muestra sus relaciones.

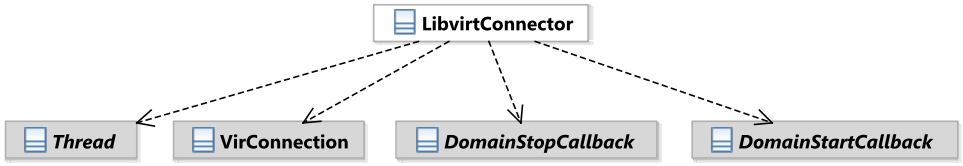

FIGURA 4.42: Relaciones de la clase `LibvirtConnector`

La clase `VirConnection` forma parte de los *bindings* de `libvirt` escritos en *Python*, y permite establecer la conexión e interactuar con el hipervisor. La conexión al hipervisor se hace con una URI, que en el caso de KVM es `qemu:///system`.

Por otra parte, toda conexión a `libvirt` tiene asociado un hilo, que muestreará periódicamente el estado del hipervisor para generar los eventos de arranque, apagado, reinicio,... de las máquinas virtuales. Estos hilos pertenecen a una clase de hilo anónima que, como todas las clases de hilo de *Python*, hereda de `Thread`.

Finalmente, las clases abstractas `DomainStartCallback` y `DomainStopCallback` definen la interfaz que se usará para procesar los eventos de arranque y apagado de una máquina virtual. El resto de eventos generados por `libvirt` se ignorarán.

4.5.6.3. La clase `DomainHandler`

Como ya hemos visto, para crear una máquina virtual es necesario

- reservar todos los recursos que tiene asignados,
- crear un fichero de definición en el que se aparezcan todos estos recursos, y
- crear la máquina virtual utilizando una llamada a `libvirt`.

Por otra parte, también es necesario procesar los eventos generados por `libvirt`. Como dijimos en la sección anterior, en el servidor de máquinas virtuales sólo se procesan dos tipos de evento:

- los eventos de arranque de una máquina virtual. Cuando se generan, la máquina virtual acaba de arrancar. Para procesarlos, se envían al propietario de la máquina virtual los datos de conexión al servidor VNC asociado a dicha máquina.
- los eventos de apagado de una máquina virtual. Cuando se generan, la máquina virtual acaba de apagarse. Para procesarlos, es necesario liberar los recursos asociados a la máquina virtual y, adelantando algo de lo que veremos más adelante, transferir las imágenes de disco al repositorio de imágenes en algunos casos.

Para acotar el nivel de complejidad de las clases del subsistema servidor de máquinas virtuales, lo más conveniente es ocultar al exterior

- los procesos configuración, creación y destrucción de las redes virtuales,
- los procesos de reserva y liberación de los recursos asignados a las máquinas virtuales,
- las interacciones con `libvirt`, y
- el procesamiento de los eventos generados por `libvirt`.

Para ello, utilizamos la clase `DomainHandler`. Sus relaciones aparecen en el diagrama de clases de la figura 4.43.

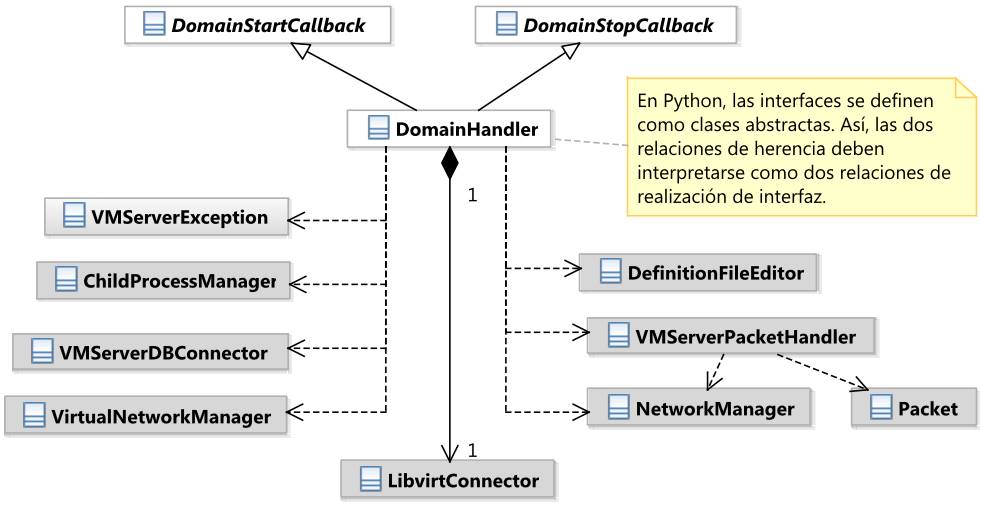

FIGURA 4.43: Relaciones de la clase `DomainHandler`

En primer lugar, la clase `DomainHandler` hereda de las clases `DomainStartCallback` y `DomainStopCallback`, por lo que podrá procesar los eventos de arranque y apagado de una máquina virtual. Asimismo, la clase `DomainHandler` utiliza un objeto `LibvirtConnector` para realizar las llamadas a `libvirt`.

Es importante notar que, en *Python*, no existen interfaces, por lo que estas deben definirse como clases abstractas. Por tanto, para implementar varias interfaces es necesario heredar de varias clases. Así, las dos relaciones de herencia del diagrama deben interpretarse como dos relaciones de realización de interfaz.

Además, como vimos en las secciones 4.4.16 y 4.4.18, cada máquina virtual tiene asociadas dos imágenes de disco `qcow2`, y una de ellas es del tipo *copy-on-write*. Durante el proceso de arranque de una máquina virtual, es necesario crear la imagen de disco *copy-on-write* y crear una copia de la imagen de disco restante. Para ello, los objetos `DomainHandler` ejecutan los comandos correspondientes utilizando los métodos de la clase `ChildProcessManager`.

Por otra parte, los métodos de la clase `DefinitionFileEditor` permiten generar los ficheros de definición de las máquinas virtuales a partir de otros que sirven como plantilla. Esto simplifica su implementación.

Asimismo, la clase `VMServerPacketHandler` dispone de métodos para leer y crear los diversos tipos de paquete del servidor de máquinas virtuales. Los objetos `DomainHandler` los utilizan para generar los paquetes con los datos de conexión a los servidores VNC de las máquinas virtuales, que enviará a los propietarios de las máquinas virtuales mediante un objeto `NetworkManager`.

Finalmente, los métodos de la clase `VMServerDBConnector` manipulan la base de datos del servidor de máquinas virtuales. Los objetos `DomainHandler` los utilizan para registrar y liberar los recursos asignados a las distintas máquinas virtuales.

4.5.6.4. Creación y edición de imágenes de disco

Para crear y editar imágenes, el servidor de máquinas virtuales debe comunicarse con el repositorio de imágenes. Como dijimos en la sección 4.4.15, para ahorrar ancho de banda el servidor de máquinas virtuales y el repositorio de imágenes no intercambian directamente imágenes de disco, sino ficheros `.zip` que contienen imágenes de disco.

Asimismo, tal y como dijimos en la sección 4.4.13, los servidores de máquinas virtuales y el repositorio de imágenes utilizarán el protocolo FTP para intercambiar ficheros comprimidos.

Así pues, para que el servidor de máquinas virtuales pueda comunicarse con el repositorio de imágenes, es necesario que

- pueda generar y descomprimir ficheros `.zip`,
- pueda transferir, mediante el protocolo FTP, un fichero comprimido al repositorio de imágenes, y
- pueda recibir, mediante el protocolo FTP, un fichero comprimido desde el repositorio de imágenes.

En esta sección presentaremos las clases que intervienen en estos procesos.

4.5.6.4.1. Colas de transferencias y de compresión

En todas las comunicaciones con el repositorio de imágenes, podemos distinguir dos fases:

- el intercambio, a través del protocolo FTP, de un fichero comprimido con imágenes de disco, y
- la descompresión o la creación de un fichero comprimido con imágenes de disco

Para reducir el tiempo que tardan en realizarse estas comunicaciones es posible realizar estas fases en paralelo. Por ello, los servidores de máquinas virtuales utilizan una cola para cada una de ellas, es decir, una cola de transferencias y una cola de compresión.

Así, la cola de transferencias contiene todas las peticiones de intercambio de ficheros comprimidos con el repositorio de imágenes, y en la cola de compresión se encuentran todas las peticiones de compresión y descompresión de ficheros `.zip`.

Por otra parte, no debemos olvidar que los usuarios pueden dedicar mucho tiempo y esfuerzo a los procesos de creación y edición de imágenes de disco. Por ello, es necesario dotar de una gran robustez a esta parte de los servidores de máquinas virtuales, de modo que sólo se pierdan datos en los casos más extremos.

Para ello, la cola de transferencias y la cola de compresión tienen persistencia en disco, y su contenido se almacena en dos tablas de la base de datos. De esta manera, aunque el demonio del servidor de máquinas virtuales se cuelgue, las imágenes modificadas por los usuarios acabarán transfiriéndose normalmente al repositorio de imágenes.

4.5.6.4.2. La clase `FileTransferThread`

La clase `FileTransferThread` se corresponde con el hilo asociado a la cola de transferencias. Este hilo intercambia paquetes con el repositorio de imágenes y realiza las correspondientes transferencias FTP cuando el repositorio de imágenes se lo indica.

El diagrama de clases de la figura 4.44 muestra las principales relaciones de la clase `FileTransferThread`.

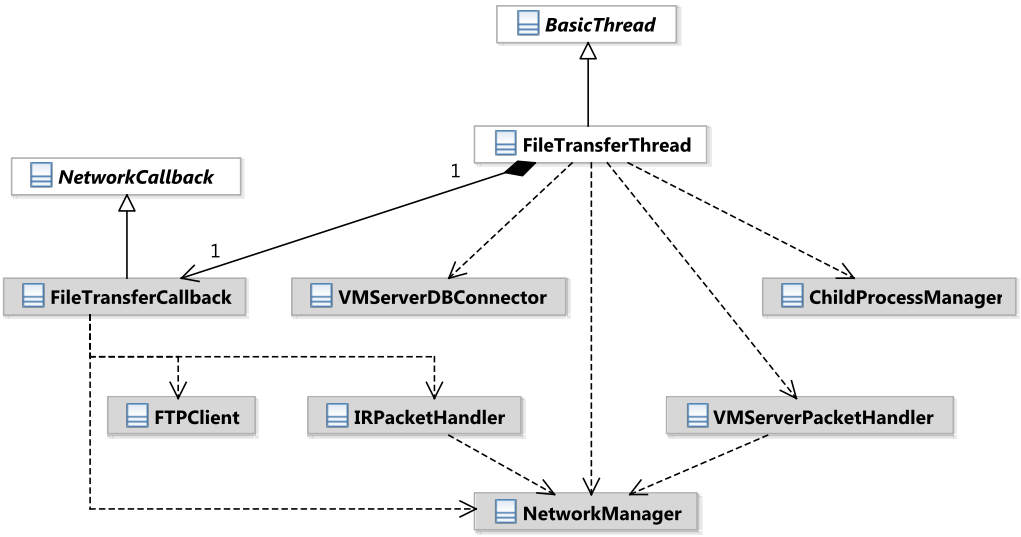

FIGURA 4.44: Principales relaciones de la clase `FileTransferThread`

Para borrar los ficheros `.zip` transferidos, la clase `FileTransferThread` utiliza los métodos de la clase `ChildProcessManager`. Asimismo, la clase `FileTransferThread` utiliza los servicios de las clases `VMServerPacketHandler` y `NetworkManager` para informar de los errores de conexión al repositorio de imágenes.

Por otra parte, los objetos `FileTransferThread` interactúan con el repositorio de imágenes a través de un objeto `FileTransferCallback`. Estos objetos intercambian paquetes con el repositorio de imágenes y realizan las transferencias FTP correspondientes. Para ello, utilizan los servicios de tres objetos: un objeto `FTPClient`, un objeto `ImageRepositoryPacketHandler` y un objeto `NetworkManager`.

Debemos recordar que, como dijimos en la sección 4.5.4.3, la clase `FTPClient` ofrece una interfaz para manipular a un alto nivel de abstracción el cliente FTP incluido en la librería estándar de *Python*.

Por otro lado, la clase `FileTransferCallback` hereda de `NetworkCallback`, ya que procesará los paquetes recibidos a través de la conexión de red con el repositorio de imágenes.

Finalmente, los elementos de la cola de transferencias se almacenan en una tabla de la base de datos del servidor de máquinas virtuales. Para desencolarlos, los objetos `FileTransferThread` utilizan un objeto `VMServerDBConnector`.

4.5.6.4.3. La clase CompressionThread

La clase CompressionThread está asociada al hilo que procesa los elementos de la cola de compresión. El diagrama de clases de la figura 4.45 muestra las principales relaciones de la clase CompressionThread.

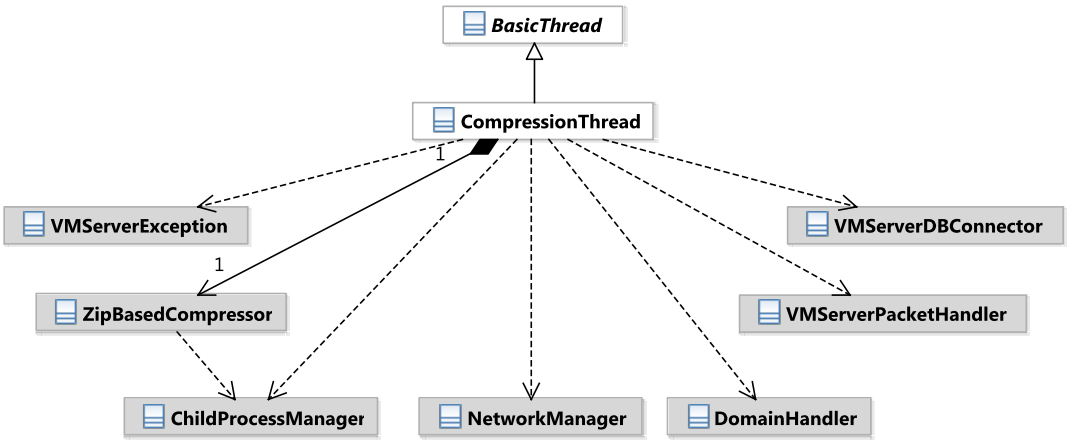

FIGURA 4.45: Principales relaciones de la clase CompressionThread

Para crear y descomprimir ficheros .zip, la clase CompressionThread utiliza los servicios de la clase ZipBasedCompressor. Esta ejecutará los comandos zip y unzip utilizando los métodos de la clase ChildProcessManager.

Aunque en la librería estándar de *Python* existen clases que permiten crear y descomprimir ficheros .zip, existe un *bug* en las mismas que impide crear archivos .zip mayores de 4 GB. Es importante notar que, cuando se comprimen imágenes de disco, frecuentemente se obtienen ficheros .zip de más de 4 GB.

Este error ha sido subsanado en las nuevas versiones de las ramas 2.x y 3.x de *Python*, pero en muchas distribuciones *Linux* actuales se sigue utilizando la versión antigua. Por ello, de cara a facilitar la instalación de *CygnusCloud*, hemos decidido no utilizar estas clases e invocar directamente a los comandos zip y unzip.

Por otra parte, la clase CompressionThread utiliza los servicios de las clases NetworkManager y VMServerPacketHandler para enviar los paquetes de error. Estos se generarán si, por ejemplo, el fichero .zip descargado del repositorio de imágenes está corrupto.

Asimismo, en algunos casos es necesario arrancar una máquina virtual tras extraer el contenido de un fichero .zip. Para ello, la clase CompressionThread utilizará los servicios de la clase DomainHandler.

Finalmente, los objetos CompressionThread utilizarán un objeto VMServerDBConnector para extraer las peticiones de compresión y descompresión de la cola de compresión (que, al igual que en el caso de la cola de transferencias, se almacenan en una tabla de la base de datos), y también para registrar y borrar las imágenes de disco en la base de datos del servidor de máquinas virtuales.

4.5.6.5. La clase VMServerReactor

La clase VMServerReactor es la clase principal del demonio del servidor de máquinas virtuales. Estos objetos se ocupan de realizar el proceso de inicialización, y también de procesar los paquetes que envía el servidor de *cluster* al servidor de máquinas virtuales a través de la conexión de control.

El diagrama de clases de la figura 4.46 muestra las principales relaciones de la clase VMServerReactor.

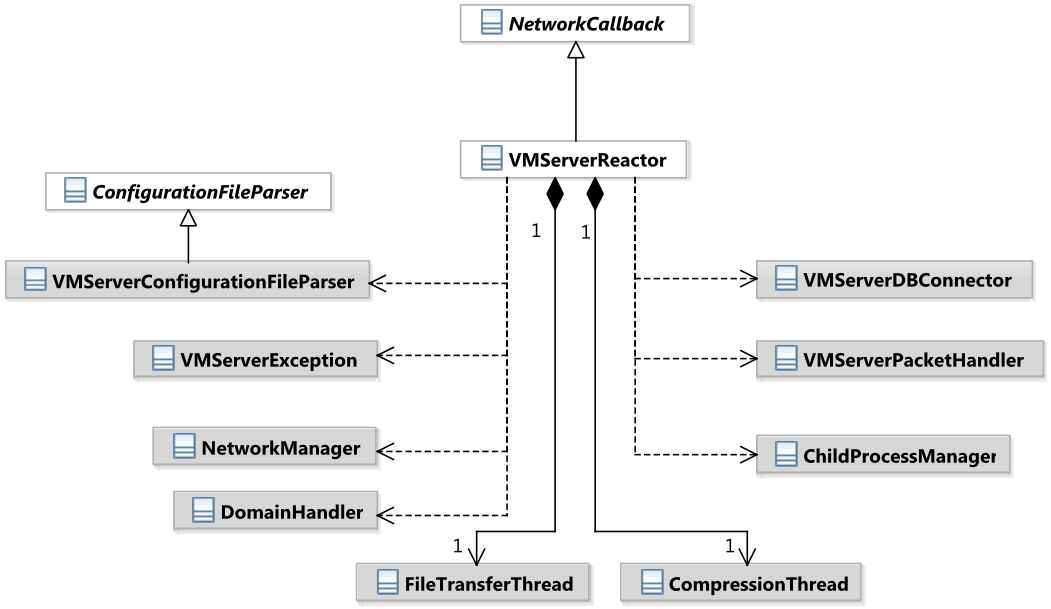

FIGURA 4.46: Principales relaciones de la clase `VMServerReactor`

En primer lugar, la clase `VMServerReactor` utiliza los servicios de la clase `VMServerConfigurationFileParser` para procesar el contenido del fichero de configuración. Como viene siendo habitual, el *parser* del fichero de configuración del servidor de máquinas virtuales, `VMServerConfigurationFileParser`, hereda de la clase abstracta `ConfigurationFileParser`, el ancestro común de todos los *parsers* de ficheros de configuración. Como ya hemos visto, esta clase está definida en el paquete `ccutils`.

Por otra parte, al igual que sucede en el caso del repositorio de imágenes, para interactuar desde el exterior con un servidor de máquinas virtuales es necesario enviarle paquetes a través de una conexión de control. A diferencia de lo que ocurría en el caso del repositorio de imágenes, sólo el servidor de *cluster* se comunicará con el servidor de máquinas virtuales.

Como la clase `VMServerReactor` hereda de `NetworkCallback`, en el servidor de máquinas virtuales se utilizará un objeto `VMServerReactor` para procesar los paquetes recibidos desde el servidor de *cluster*.

Para realizar dicho procesamiento, es necesario enviar respuestas al servidor de *cluster*, iniciar transferencias con el repositorio de imágenes y crear o destruir máquinas virtuales. Por ello, la clase `VMServerReactor` utiliza los servicios de las clases `NetworkManager`, `VMServerPacketHandler`, `DomainHandler` y `VMServerDBConnector`.

Finalmente, es importante notar que la clase `VMServerReactor` no iniciará directamente las transferencias con el repositorio de imágenes. Al recibir el paquete correspondiente, encolará una petición en la cola de transferencias. Para procesar dicha petición, intervendrán los objetos `FileTransferThread` y `CompressionThread`.

4.5.6.6. Arranque del servidor de máquinas virtuales: secuencia básica

En esta sección mostraremos el proceso de arranque del servidor de máquinas virtuales. Para ello, haremos uso del diagrama de secuencia de la figura 4.47. Por claridad, hemos supuesto que, durante el proceso de arranque, no se produce ningún error.

Como podemos observar en dicho diagrama, se siguen los siguientes pasos:

1. se realiza, si es necesario, la configuración inicial de la base de datos del servidor de máquinas virtuales.

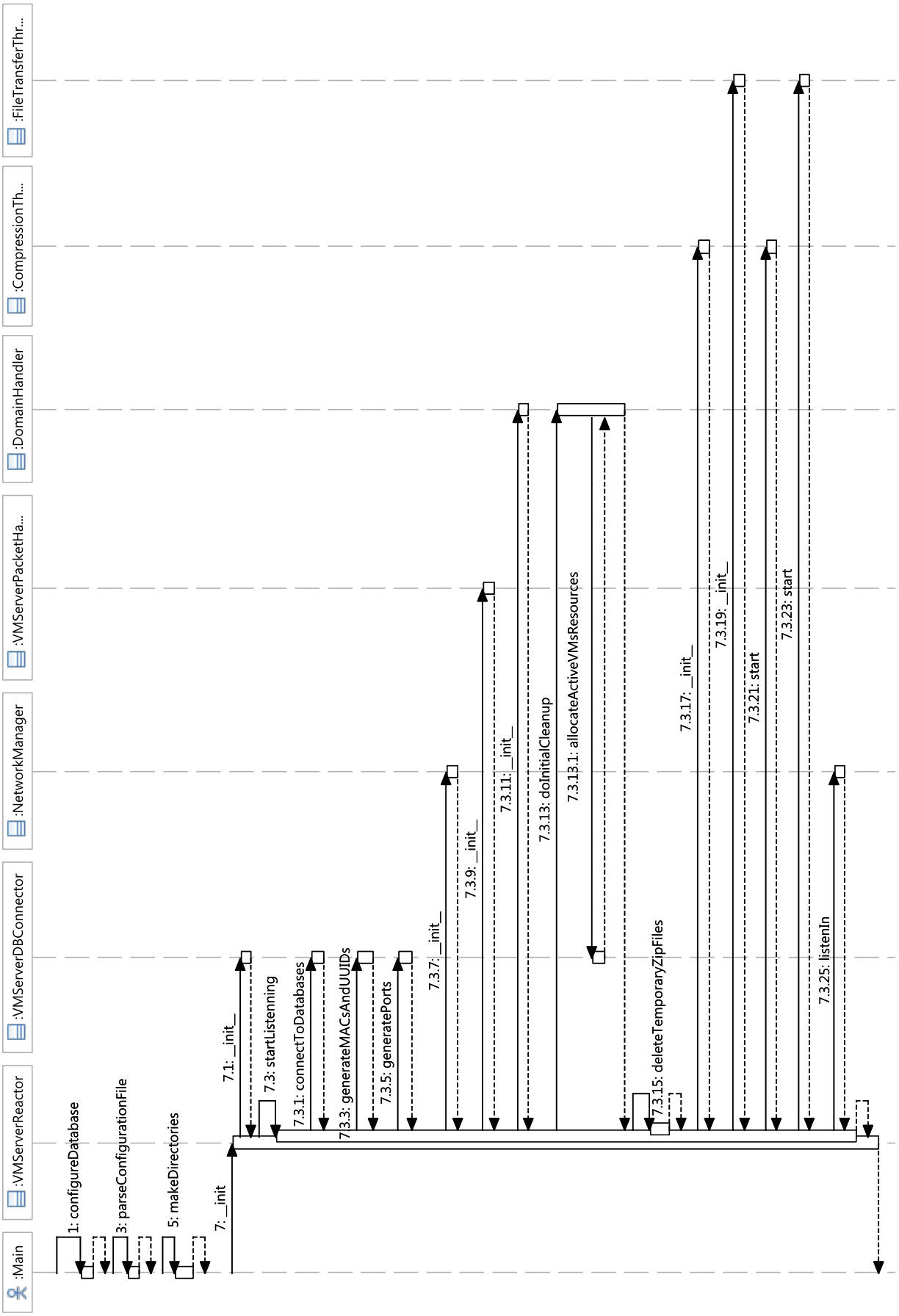

FIGURA 4.47: Arranque del servidor de máquinas virtuales: secuencia básica

2. se parsea el fichero de configuración.
3. se crean, si es necesario, los distintos directorios utilizados por el servidor de máquinas virtuales.
4. se establece la conexión con la base de datos del servidor de máquinas virtuales.
5. se crean las direcciones MAC y los UUIDs que usarán las máquinas virtuales activas. Todas las direcciones MAC creadas son de la forma

   ```
   2C:00:00:00:00:M
   ```

   siendo `M` un número entero entre 0 y 255 codificado en hexadecimal.

   Por tanto, todas las direcciones MAC son *unicast* y de uso local. Asimismo, el servidor de máquinas virtuales podrá albergar un máximo de 256 máquinas virtuales.

   Teniendo en cuenta los recursos que, como mínimo, hay que asociar a un sistema operativo con interfaz gráfica de usuario, resulta difícil alcanzar este límite con el *hardware* actual.
6. se reservan 512 puertos, que utilizarán los servidores VNC de las máquinas virtuales y los *websockets* que estos tienen asociados.

   Por convención, los puertos pares se asignarán a los servidores VNC, y los puertos impares consecutivos a estos se asignarán a los *websockets*.
7. se prepara todo lo necesario para atender las peticiones de los usuarios.
8. se comprueba qué máquinas virtuales activas están registradas en la base de datos, reasignando o liberando recursos como corresponda. Este paso permite que, si el demonio del servidor de máquinas virtuales termina de forma abrupta,
   - las máquinas virtuales activas vuelvan a reasociarse al servidor de máquinas virtuales, y que
   - los recursos asociados a las máquinas virtuales apagadas se liberen.
9. se borran los ficheros `.zip` parcialmente generados (si los hay).
10. se inicializan los hilos de procesamiento asociados a la cola de transferencias y a la cola de compresión.
11. se crea la conexión de control. A partir de este momento, el servidor de máquinas virtuales estará preparado para atender las peticiones del servidor de *cluster*.

4.5.6.7. Arranque del servidor de máquinas virtuales: tratamiento de errores

Durante el proceso de arranque del demonio del servidor de máquinas virtuales, se producirán errores cuando

- el contenido del fichero de configuración es incorrecto.
- la interfaz de red por la que circulará el tráfico VNC no está lista.
- el puerto de escucha ya está en uso.
- no es posible inicializar la base de datos del servidor de máquinas virtuales.

Sea cual sea el error detectado, desde el hilo principal

1. se destruyen todas las máquinas virtuales activas.
2. si están iniciados, se paran los hilos que procesan los elementos de la cola de transferencias y de la cola de compresión.

3. se cierra la conexión de control y se detiene el servicio de red. A partir de este momento, el servidor de máquinas virtuales dejará de atender peticiones.

Nuevamente, y por las mismas razones que comentamos en la sección 4.5.5.7, el servicio de red deberá detenerse desde el hilo principal para evitar que se produzca un cuelgue.

4.5.6.8. Interacciones básicas con el servidor de *cluster*

En esta sección, mostraremos todas las interacciones básicas que pueden tener lugar entre un servidor de máquinas virtuales y el servidor de *cluster*.

Por claridad, en todos los casos comenzaremos mostrando el flujo básico, para posteriormente mostrar los flujos alternativos, que se corresponden con el tratamiento de diversos errores.

Asimismo, en esta sección omitiremos las interacciones que despliegan, crean, editan o borran imágenes de disco: debido a su enorme entidad, preferimos tratarlas en secciones separadas.

4.5.6.8.1. Solicitud de estado

Para poder realizar correctamente el balanceado de carga entre los diversos servidores de máquinas virtuales, los servidores de *cluster* deben conocer, en tiempo real, el estado de cada servidor de máquinas virtuales.

Para averiguar el estado de un servidor de máquinas virtuales, el servidor de *cluster* le enviará un paquete del tipo Solicitud de Estado. Estos paquetes se procesan en el servidor de máquinas virtuales como se indica en la figura 4.48.

Como puede observarse en el diagrama, se siguen los siguientes pasos:

1. el servidor de *cluster* envía un paquete del tipo Solicitud de Estado al servidor de máquinas virtuales. Este no contiene ningún tipo de información adicional.
2. el demonio del servidor de máquinas virtuales lee el contenido del paquete. Acto seguido,
   a) averigua las cantidades de memoria RAM en uso y disponible. Para ello, ejecuta el comando `free`.
   b) averigua el número de CPUs virtuales en uso consultando los recursos asignados a las máquinas virtuales activas.
   c) utiliza los métodos del módulo `os` de la librería estándar de *Python* para averiguar el espacio en disco utilizado y el espacio en disco disponible.
   d) utiliza la utilidad `qemu-img info` para averiguar el espacio en disco adicional que, como máximo, pueden requerir todas las máquinas virtuales activas.
   Este paso es fundamental: como las imágenes de disco `qcow2` soportan compresión, se pueden expandir a medida que los usuarios trabajan con las máquinas virtuales y, por tanto, ocupar más espacio en disco.
3. se construye un paquete del tipo Estado del servidor de máquinas virtuales con toda la información recopilada. Acto seguido, el servidor de máquinas virtuales envía dicho paquete al servidor de *cluster*.

4.5.6.8.2. Creación de una máquina virtual: secuencia básica

Los servidores de máquinas virtuales sólo crean una máquina virtual bajo petición expresa del servidor de *cluster*.

Para enviar esas peticiones, el servidor de *cluster* utiliza un paquete del tipo Creación de máquina virtual, que se procesa de acuerdo al diagrama de secuencia de la figura 4.49. Por claridad, en dicho diagrama sólo hemos recogido la secuencia básica, en la que no aparecen errores.

Como podemos observar en el diagrama de secuencia de la figura 4.49, en el arranque de una máquina virtual se siguen estos pasos:

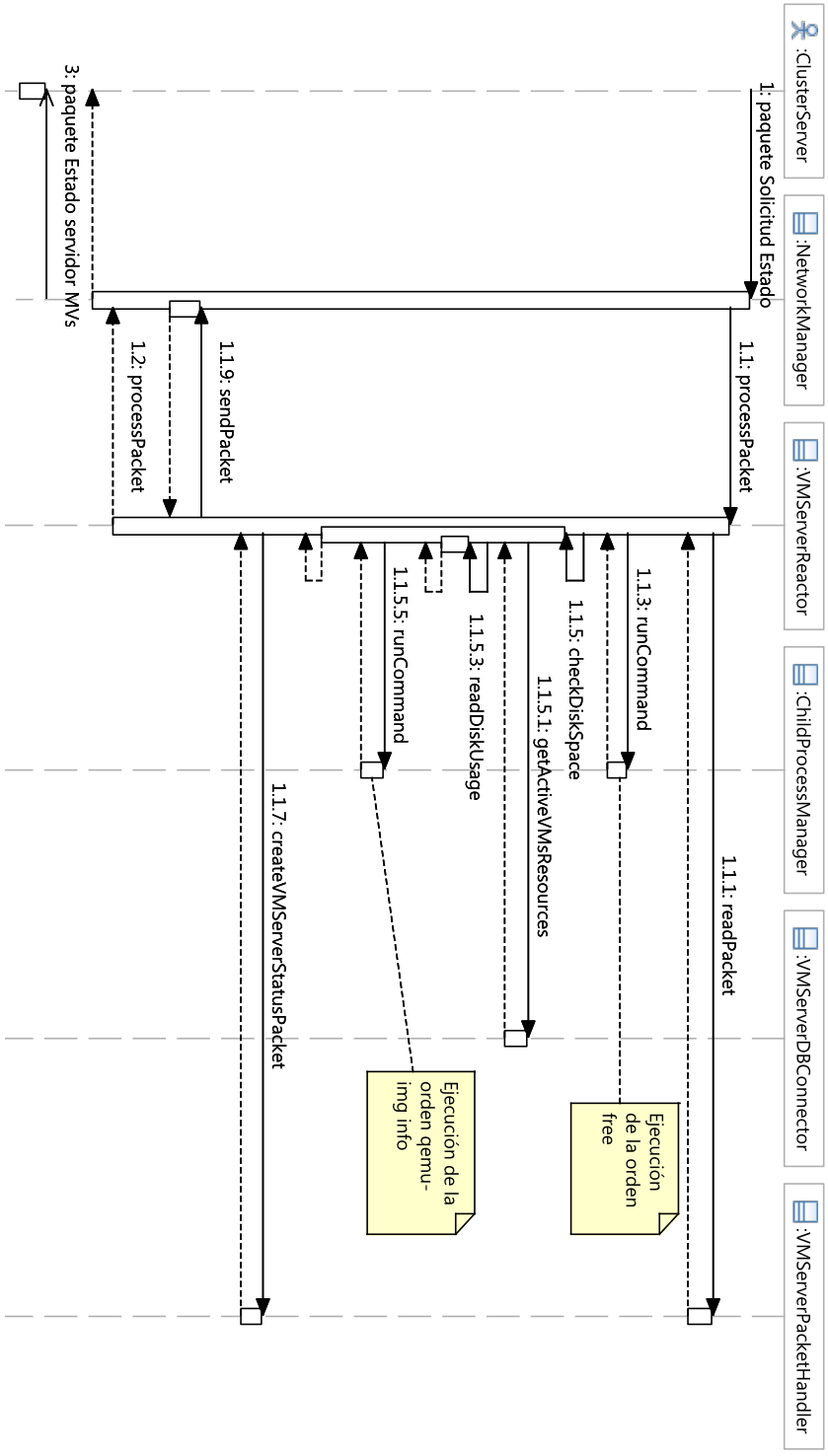

FIGURA 4.48: Solicitud del estado del servidor de máquinas virtuales

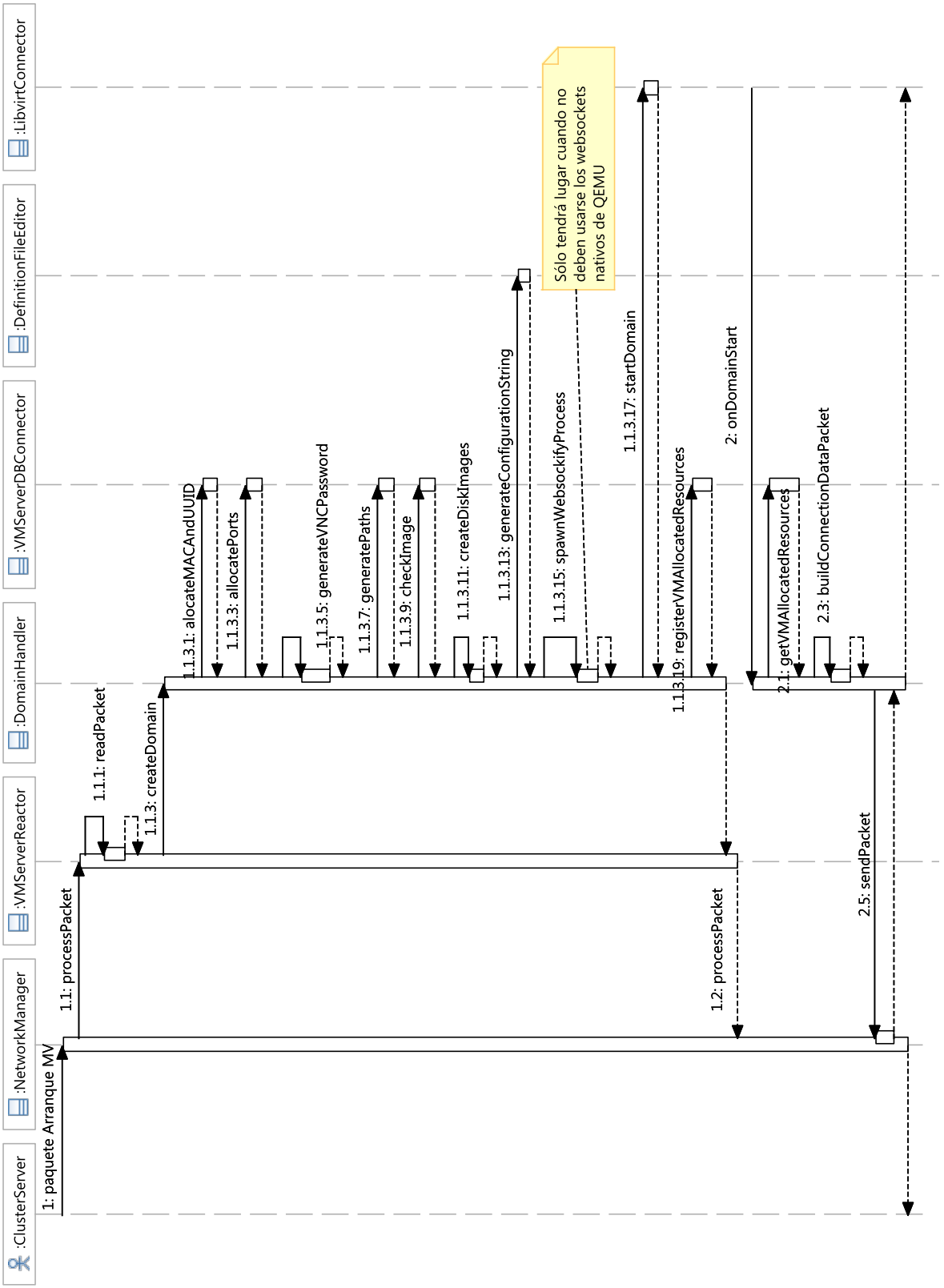

FIGURA 4.49: Creación de una máquina virtual: secuencia básica

1. se lee el paquete del tipo Creación de máquina virtual. Estos paquetes contienen el identificador único de la petición, el identificador del usuario que la ha enviado y el identificador de la imagen que utilizará la nueva máquina virtual que se creará.

2. se reservan los recursos asignados a la máquina virtual. Para ello,

   a) se reservan una dirección MAC y un UUID para la máquina virtual.

   b) se reservan dos puertos *consecutivos* para la máquina virtual. Como ya hemos dicho, el puerto par se asignará al servidor VNC de la máquina virtual, y el puerto impar se asignará al *websocket*.

   c) se genera aleatoriamente una contraseña para el servidor VNC. Para ello, se ejecuta el comando `openssl`. La longitud de la contraseña generada se fija en el fichero de configuración del demonio del servidor de máquinas virtuales.

   d) se crea la imagen de disco *copy-on-write* y la imagen de disco que almacenará los datos temporales y el fichero de paginación. Para ello, se ejecutarán las órdenes `qemu-img create` y `cp`.

3. se arranca la máquina virtual. Para ello,

   a) genera el fichero de definición de la máquina virtual. Por motivos de eficiencia, su contenido no se escribe a disco, sino que se genera un *string* con el contenido de dicho fichero.

   b) si el servidor de máquinas virtuales está configurado para no utilizar los *websockets* nativos de QEMU, se arranca el proceso `websockify`, que se ejecutará en *background*.

   c) se solicita a `libvirt` la creación de una nueva máquina virtual.

4. se registran los recursos asignados a la máquina virtual en la base de datos del servidor de máquinas virtuales.

5. cuando se trata el evento de arranque de máquina virtual generado por `libvirt`,

   a) se leen de la base de datos del servidor de máquinas virtuales el puerto del *websocket*, la contraseña del servidor VNC, el identificador único de la petición y el identificador del usuario que ha enviado la petición de arranque de la máquina virtual.

   b) se construye un paquete del tipo Datos de conexión de máquina virtual. Este contiene la información que se ha extraído de la base de datos en el paso anterior.

   c) se envía dicho paquete al servidor de *cluster*.

El hecho de que las direcciones MAC libres se extraigan de la base de datos no es casual. Siempre que el servidor DHCP concede una dirección IP a una máquina virtual lo hace durante un tiempo prefijado.

Así, cuando se arranca y detiene un gran número de máquinas virtuales en un corto periodo de tiempo, los tiempos de concesión de las direcciones IP no habrán expirado, por lo que corremos el riesgo de el servidor DHCP se quede sin direcciones IP para asignar.

Para evitar la aparición de este problema, podemos aprovechar el hecho de que el protocolo DHCP asigna direcciones IP a direcciones MAC: puesto que estas siempre se extraen de la base de datos, las nuevas máquinas virtuales podrán reutilizar las direcciones IP de las máquinas antiguas y, mientras el servidor de máquinas virtuales no aloje el número máximo de máquinas virtuales, no habrá problemas.

4.5.6.8.3. Creación de una máquina virtual: tratamiento de errores

En el proceso de arranque de una máquina virtual, se producirán errores cuando

- la conexión con la base de datos falla, cuando
- falla la llamada a `libvirt`, o cuando
- la petición de arranque es incorrecta. Esto sucederá si
  - el servidor de máquinas virtuales no dispone de la imagen que es necesario utilizar.
  - las imágenes de disco asociadas a la máquina virtual se están editando en el servidor de máquinas virtuales, o cuando
  - el servidor de máquinas virtuales no disponga de suficientes CPUs, memoria RAM o espacio libre en disco.

Estos errores pueden detectarse en dos puntos de la secuencia:

- al consultar la base de datos del servidor de máquinas virtuales para obtener el estado de la imagen.
- al realizar la llamada a `libvirt`.

Todos ellos, salvo los errores en las llamadas a `libvirt`, se detectan en el primer punto de la secuencia. No obstante, con independencia del punto en el que se detecten, todos se tratan de la misma manera.

1. se liberan todos los recursos que hayan sido asignados a la máquina virtual. Por ejemplo, si las imágenes de disco no se han creado cuando se produce el error, no se intentarán liberar.
2. se aborta el proceso de arranque de forma inmediata.

El diagrama de secuencia de la figura 4.50 muestra cómo se procede cuando se intenta utilizar una imagen que no existe en el servidor de máquinas virtuales para crear una máquina virtual.

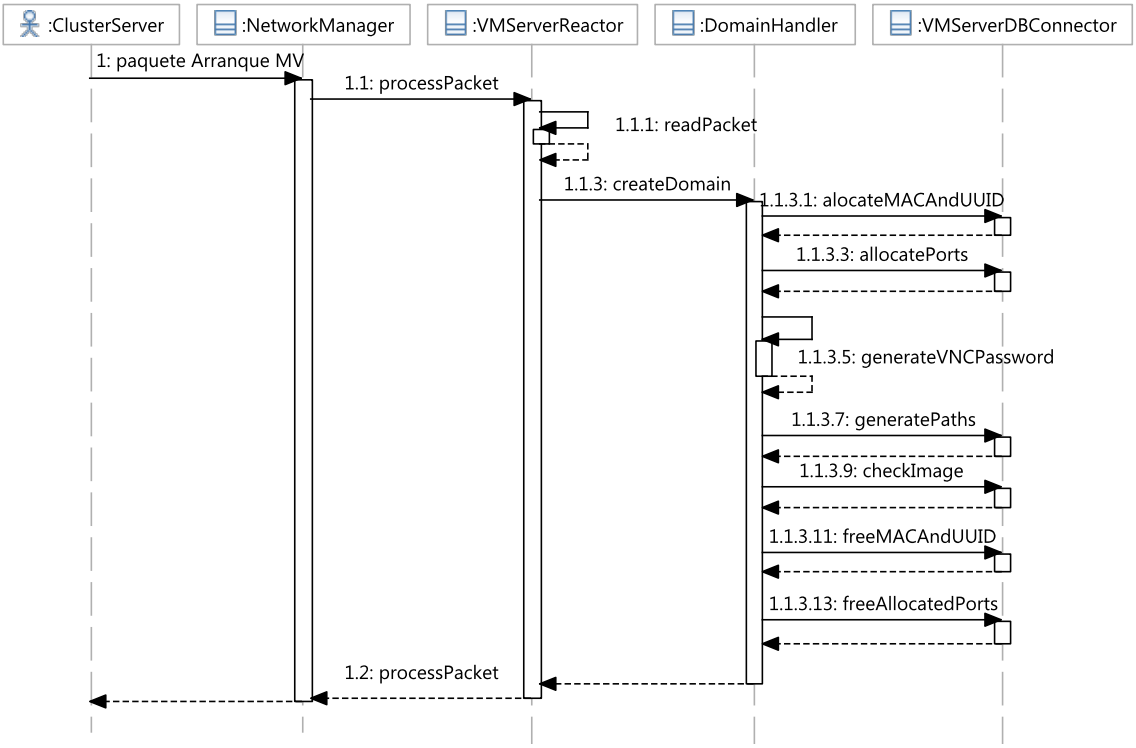

FIGURA 4.50: Arranque de una máquina virtual: tratamiento de errores

Como el lector habrá observado ya, si falla una petición de arranque de máquina virtual no se generará ningún paquete de error. Esto nos permite ahorrar ancho de banda, ya que

- el servidor de *cluster* nunca enviará peticiones que pueden fallar a los servidores de máquinas virtuales, y
- las llamadas a `libvirt` sólo fallan en casos realmente excepcionales.

4.5.6.8.4. Apagado de una máquina virtual

Cuando se apaga una máquina virtual, se liberan todos los recursos que esta tiene asociados. Este proceso se muestra en el diagrama de secuencia de la figura 4.51.

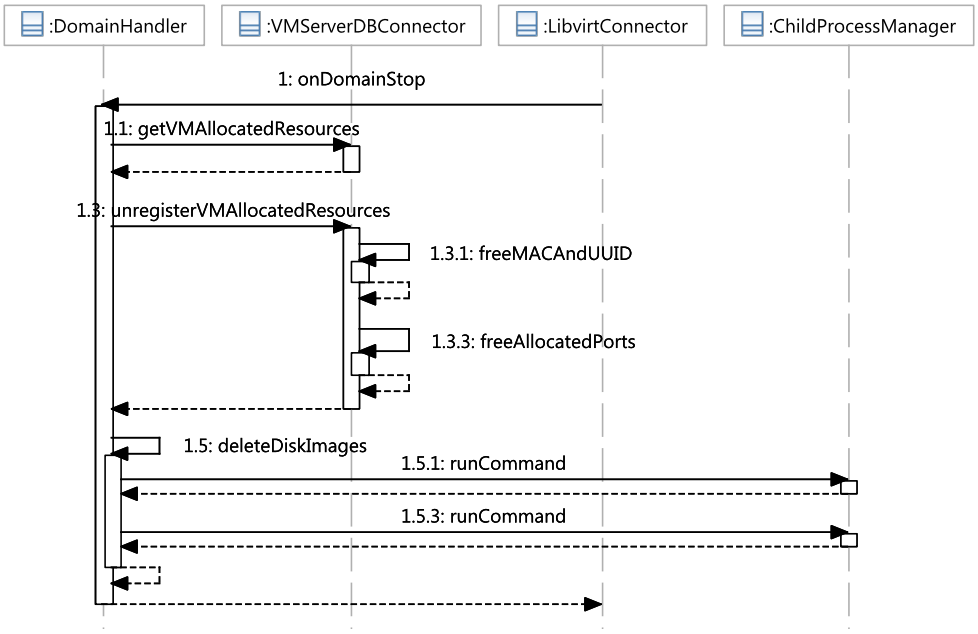

FIGURA 4.51: Apagado de una máquina virtual: secuencia básica

Como podemos observar en dicho diagrama, sucede lo siguiente:

1. cuando la máquina virtual se apaga, `libvirt` genera el evento correspondiente.
2. para procesar ese evento,
   a) se averigua cuáles son los recursos asignados a la máquina virtual que se ha apagado. Para ello, se realiza una consulta en la base de datos.
   b) se liberan la dirección MAC, el UUID y los puertos asignados al servidor VNC y al *websocket*. También se borran las imágenes de disco ejecutando el comando `rm`.

Cuando la máquina virtual se apaga, `libvirt` se ocupará de destruir el servidor VNC que tiene asociado, por lo que no debemos preocuparnos por él.
Por otra parte,

- si utilizamos los *websockets* nativos de QEMU, libvirt también se ocupará de destruirlos tras apagar el servidor VNC.
- si no utilizamos los *websockets* nativos de QEMU, el proceso `websockify` detectará el apagado del servidor VNC, y responderá a este apagándose. Por ello, no es necesario matarlo.

Finalmente, como que las dos imágenes de disco que utilizaba la máquina virtual se borran, cuando esta se apaga se destruyen todos los datos que se hayan escrito en disco durante su ejecución.

4.5.6.8.5. Solicitud de datos de conexión

Los usuarios de las máquinas virtuales pueden cerrar el cliente VNC que utilizan de forma accidental. Para que los usuarios puedan seguir utilizando la máquina virtual en estos casos, es necesario que los servidores de máquinas virtuales suministren al servidor de *cluster* los datos de conexión del servidor VNC asociado a cada máquina virtual activa.

Para solicitar estos datos, el servidor de *cluster* envía a los servidores de máquinas virtuales un paquete del tipo Solicitud de datos de conexión VNC. Este paquete no contiene información adicional, y se procesa de acuerdo al diagrama de secuencia de la figura 4.52.

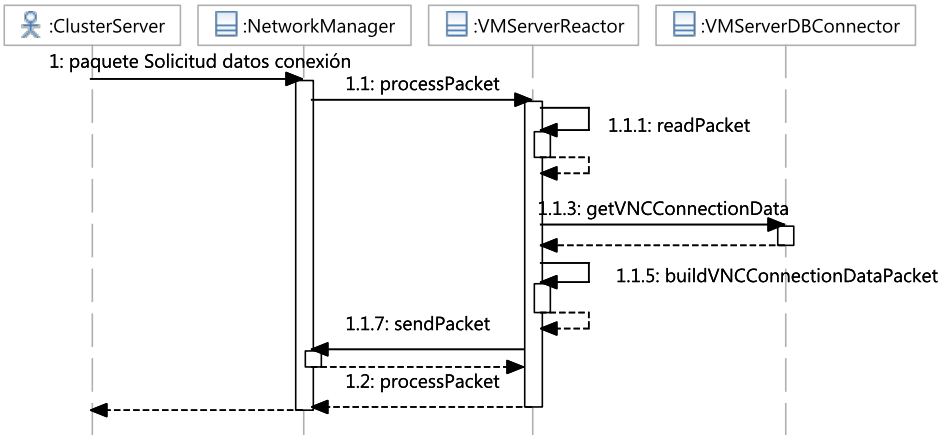

FIGURA 4.52: Envío de los datos de conexión de los servidores VNC

Como podemos observar, tras recibir el paquete en el servidor de máquinas virtuales se siguen los siguientes pasos:

1. se lee el contenido de ese paquete
2. se extraen de la base de datos los datos de conexión de los servidores VNC asociados a las máquinas virtuales activas.
3. se construye un paquete del tipo Datos de conexión VNC. Estos paquetes contienen la dirección IP del servidor de máquinas virtuales, los puertos de los *websockets* y las contraseñas de todos los servidores VNC activos.
4. se envía dicho paquete al servidor de *cluster*.

4.5.6.8.6. Solicitud de los identificadores únicos de las máquinas virtuales activas

Cada máquina virtual puede identificarse de forma única utilizando el identificador de la petición que la creó. De ahora en adelante, también llamaremos a dicho identificador identificador único de la máquina virtual.

Para conocer la distribución de las máquinas virtuales activas, el servidor de *cluster* envía a cada servidor de máquinas virtuales un paquete del tipo Solicitud de IDs de máquinas virtuales de máquinas activas.

Como esta interacción es muy similar a la anterior, no volveremos a reproducirla en esta sección.

No obstante, debemos notar que, tras enviar la consulta a la base de datos, el servidor de máquinas virtuales creará y enviará un paquete del tipo IDs de máquinas virtuales al servidor de *cluster*.

Estos paquetes contienen, como su nombre indica, los identificadores de todas las máquinas virtuales activas que alberga el servidor de *cluster*.

4.5.6.8.7. Reinicio forzoso de una máquina virtual

Cuando una máquina virtual se cuelga, el usuario que la utiliza podrá solicitar el reinicio forzoso de la misma.

Para ello, el servidor de *cluster* enviará al servidor de máquinas virtuales un paquete del tipo Reiniciar máquina virtual, que contiene el identificador único de la máquina virtual a reiniciar.

Para procesarlo, en el servidor de máquinas virtuales se seguirán los siguientes pasos:

1. se lee el contenido del paquete.
2. se reinicia la máquina virtual mediante una llamada a `libvirt`.

Es importante notar que, en caso de que el servidor de máquinas virtuales no aloje la máquina virtual, la petición se ignorará. Esto sólo ocurre en contadas ocasiones, y al no enviar un paquete de error podemos ahorrar ancho de banda.

4.5.6.8.8. Apagado del servidor de máquinas virtuales

Para apagar un servidor de máquinas virtuales, el servidor de *cluster* le enviará un paquete del tipo Apagado inmediato o un paquete del tipo Apagado con espera. Los primeros no contienen información adicional, y los segundos contienen un intervalo de espera en segundos.

El diagrama de secuencia de la figura 4.53 muestra el procesamiento de un paquete del tipo Apagado con espera, que consta de los siguientes pasos:

1. tras leer le contenido del paquete, se esperará durante el tiempo que se indica en el paquete. Es importante notar que, durante el tiempo de espera, el servidor de máquinas virtuales dejará de atender nuevas peticiones.

   Con esta espera, se pretende dar tiempo a los usuarios para que apaguen sus máquinas virtuales. Por ello, tan pronto como no haya ninguna máquina virtual activa esta espera finalizará.

2. tras finalizar la espera, el hilo principal invoca a la rutina de apagado. Durante la ejecución de ese método,

   a) se apagan todas las máquinas virtuales activas. Sus recursos se liberarán como vimos en la sección 4.5.6.9.6.

   b) se paran los hilos asociados a la cola de transferencia y a la cola de compresión. Puesto que estas colas tienen persistencia en disco, el procesamiento de sus elementos se reanudará en el siguiente arranque del servidor de máquinas virtuales, y no se perderán datos.

   c) se cierra la conexión de control y se detiene el servicio de red

Por otra parte, los paquetes del tipo Apagado inmediato se procesan de forma muy similar. No obstante, al procesarlos el tiempo de espera será siempre 0, por lo que nunca se esperará a que los usuarios apaguen las máquinas virtuales activas.

Finalmente, el servicio de red deberá detenerse desde el hilo principal para evitar que se produzca un cuelgue. Justificamos esto en detalle en la sección 4.5.5.7.

4.5.6.9. Creación y edición de imágenes de disco

En esta sección mostraremos cómo los servidores de máquinas virtuales interactúan con el servidor de *cluster* y el repositorio de imágenes durante los procesos de creación y edición de imágenes.

Nuevamente, en todos los casos empezaremos mostrando el flujo básico. Posteriormente, mostraremos los distintos flujos alternativos, que están relacionados con el tratamiento de errores.

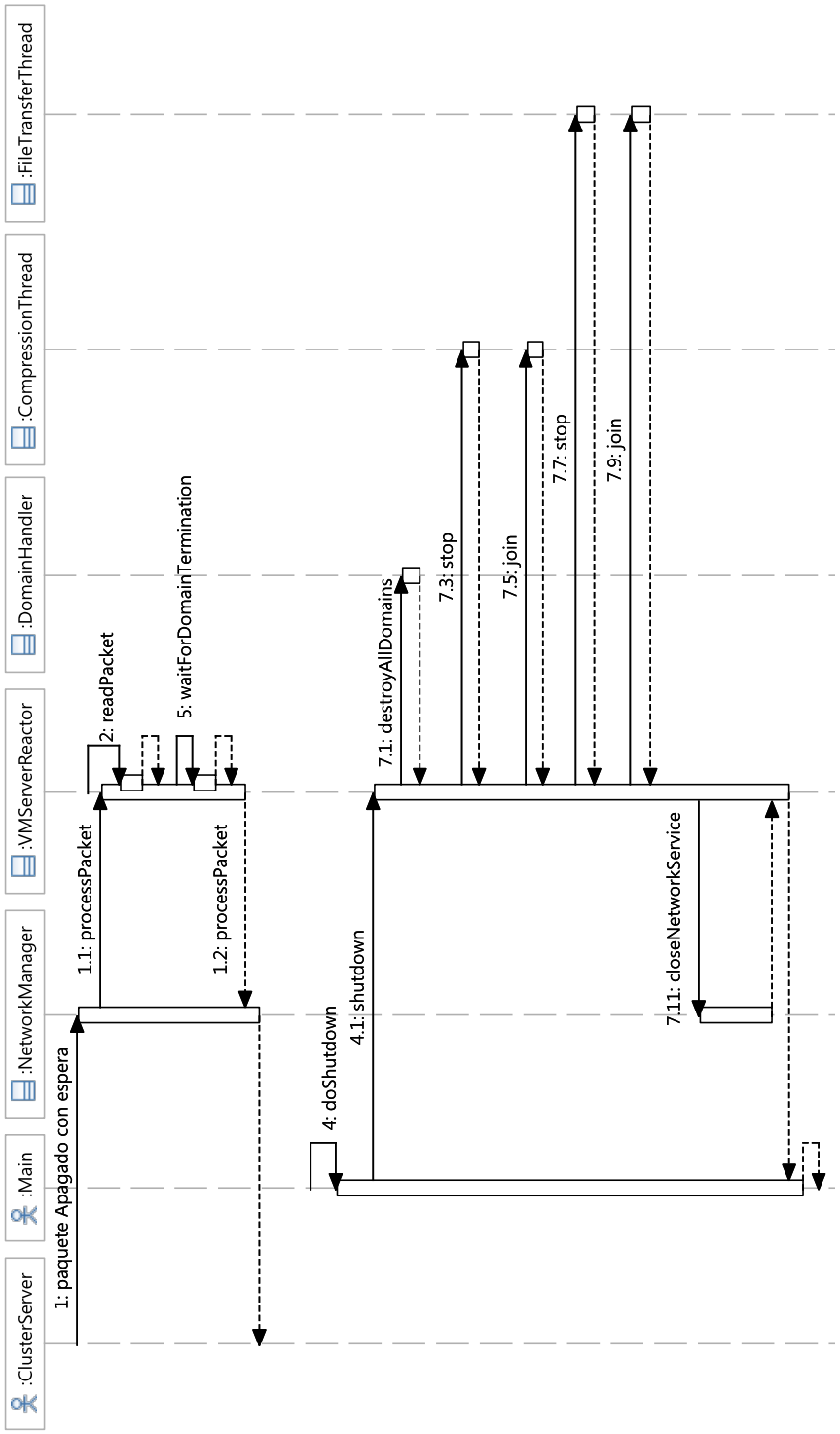

FIGURA 4.53: Apagado del servidor de máquinas virtuales

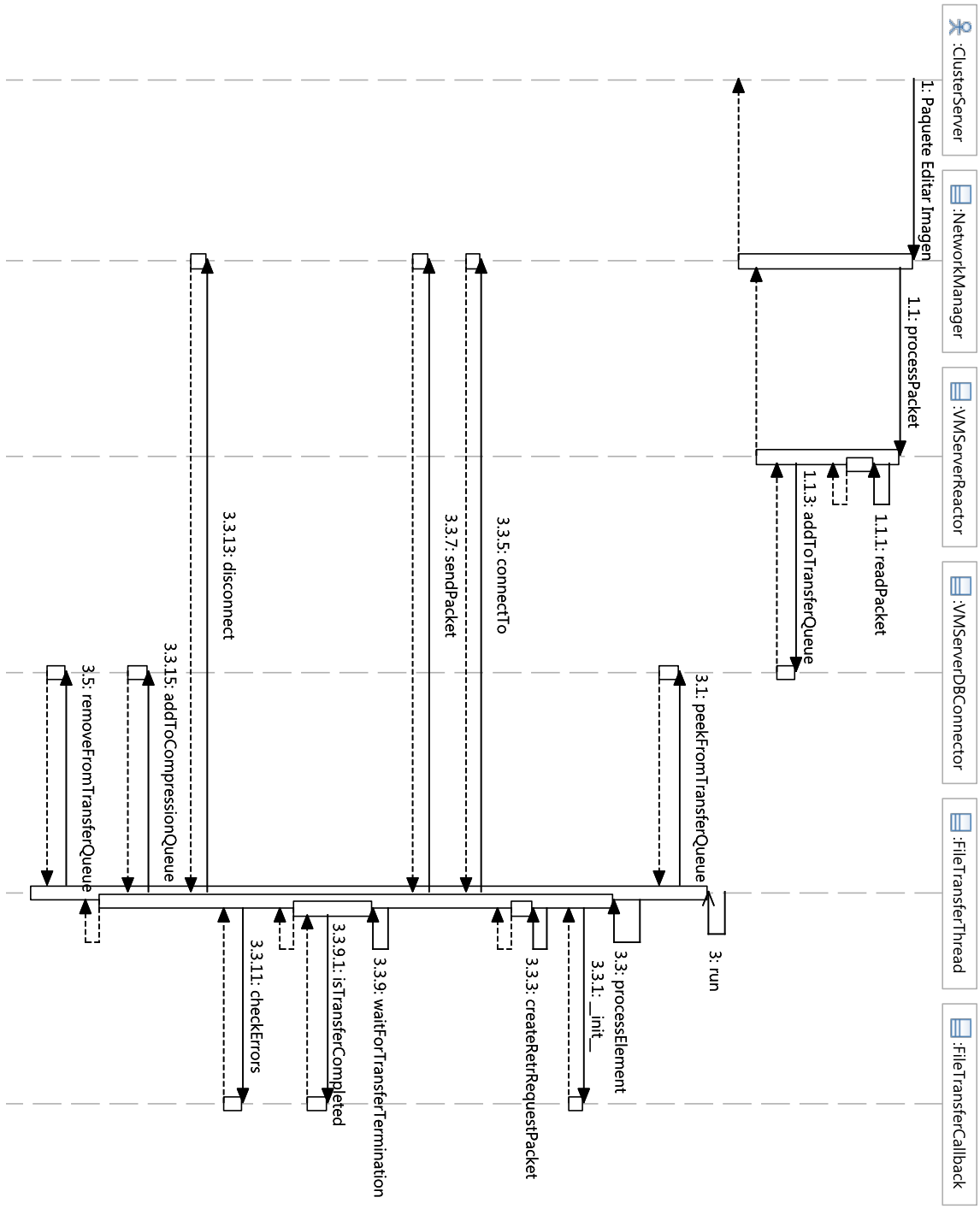

FIGURA 4.54: Creación y edición de imágenes de disco: descarga del fichero comprimido desde el repositorio de imágenes

4.5.6.9.1. Descarga del fichero comprimido

Para empezar, nos centraremos en el proceso de descarga del fichero comprimido desde el repositorio de imágenes, que se recoge en el diagrama de secuencia de la figura 4.54. En la interacción que recoge dicho diagrama no se produce ningún error, y se siguen estos pasos:

1. el servidor de *cluster* envía al servidor de máquinas virtuales un paquete del tipo Editar imagen. Este contiene
   - la dirección IP y el puerto de la conexión de control del repositorio de imágenes,
   - el identificador único del fichero comprimido a descargar.
   - un *flag* que indica si se va a crear una imagen nueva a partir de la imagen descargada o se va a editar la imagen descargada.
   - el identificador único del usuario que ha generado la petición.
   - el identificador único de la petición.
2. tras leer el paquete, se crea y añade la petición correspondiente a la cola de transferencias.
3. cuando le llega el turno, la petición se procesará en el hilo de transferencias. Para ello,
   a) se establecerá la conexión con el repositorio
   b) se iniciará el diálogo con el repositorio de imágenes para obtener el fichero comprimido. Para ello, se enviará un paquete del tipo Petición `FTP RETR` al repositorio de imágenes.
   Puesto que el repositorio de imágenes se ocupa de gestionar los identificadores de las imágenes, cuando se crea una imagen nueva deberá solicitarse un nuevo identificador de imagen al repositorio de imágenes.
4. cuando finaliza la transferencia FTP,
   a) se cerrará la conexión con el repositorio de imágenes, y
   b) se construirá y añadirá una petición a la cola de compresión.

Por simplicidad, las peticiones de transferencia de ficheros se procesarán de una en una.

Asimismo, como el número de servidores de máquinas virtuales del *cluster* puede ser muy grande, estos se desconectan del repositorio de imágenes tan pronto como finaliza la transferencia para evitar, en la medida de lo posible, que el repositorio de imágenes se sature.

Por otra parte, como para descargar la imagen se utiliza el identificador de cualquier imagen existente, será posible crear o editar imágenes de disco en varias fases. Para ello, bastará con repetir varias veces el proceso que mostramos en esta sección y en las siguientes.

Finalmente, los servidores de *cluster* siempre suministran a los servidores de máquinas virtuales los datos de conexión al repositorio de imágenes. Aunque con ello se consume un poco más de ancho de banda, se gana en flexibilidad, ya que es posible utilizar varios repositorios de imágenes en el mismo *cluster*.

4.5.6.9.2. Errores durante la descarga del fichero comprimido

Durante la descarga del fichero comprimido, se producirán errores cuando

- el repositorio de imágenes no tiene el fichero comprimido que se pretende descargar, cuando
- no se puede establecer la comunicación con el repositorio de imágenes, o cuando
- la transferencia FTP con el repositorio de imágenes falla.

Centrémonos en el primer error. Puesto que las transferencias con el repositorio de imágenes no se inician inmediatamente, es posible que ocurra lo siguiente:

1. un profesor decide crear una máquina virtual a partir de las imágenes de disco asociadas al identificador 1.

2. mientras la petición de transferencia está encolada, un administrador borra el fichero comprimido del repositorio de imágenes.

3. cuando se va a tratar la petición de transferencia, las imágenes de disco asociadas al identificador 1 han dejado de existir en el repositorio de imágenes.

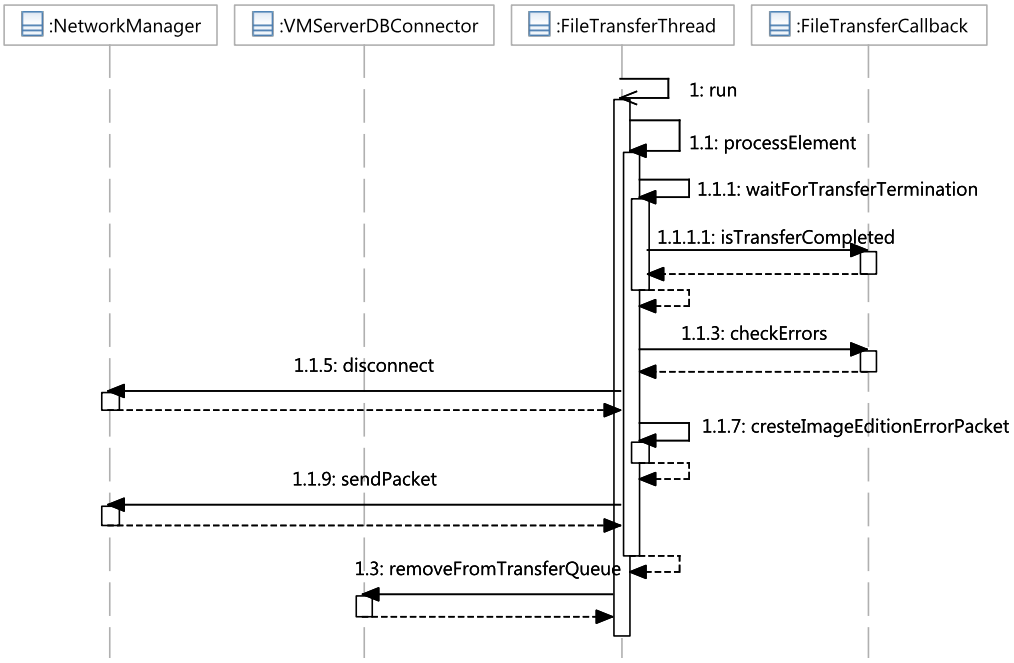

FIGURA 4.55: Creación y edición de imágenes de disco: imágenes no registradas en el repositorio de imágenes

El diagrama de secuencia de la figura 4.55 muestra cómo se tratan los errores del primer tipo. Por claridad, en él hemos omitido los pasos previos a la detección del error, que son exactamente iguales que en la secuencia básica.

Como podemos observar en dicho diagrama, tras detectar el error

1. se cierra la conexión con el repositorio de imágenes.

2. se construye un paquete del tipo Error de edición de imagen. Estos contienen el identificador de la petición y un código de descripción del error.

3. se envía dicho paquete de error al servidor de *cluster*.

4. la petición no se inserta en la cola de compresión, por lo que su procesamiento termina inmediatamente.

Por otra parte, el diagrama de secuencia de la figura 4.56 muestra cómo se tratan los errores de conexión al repositorio de imágenes. Nuevamente, en él hemos omitido los pasos previos a la detección del error, que son idénticos a los de la secuencia básica.

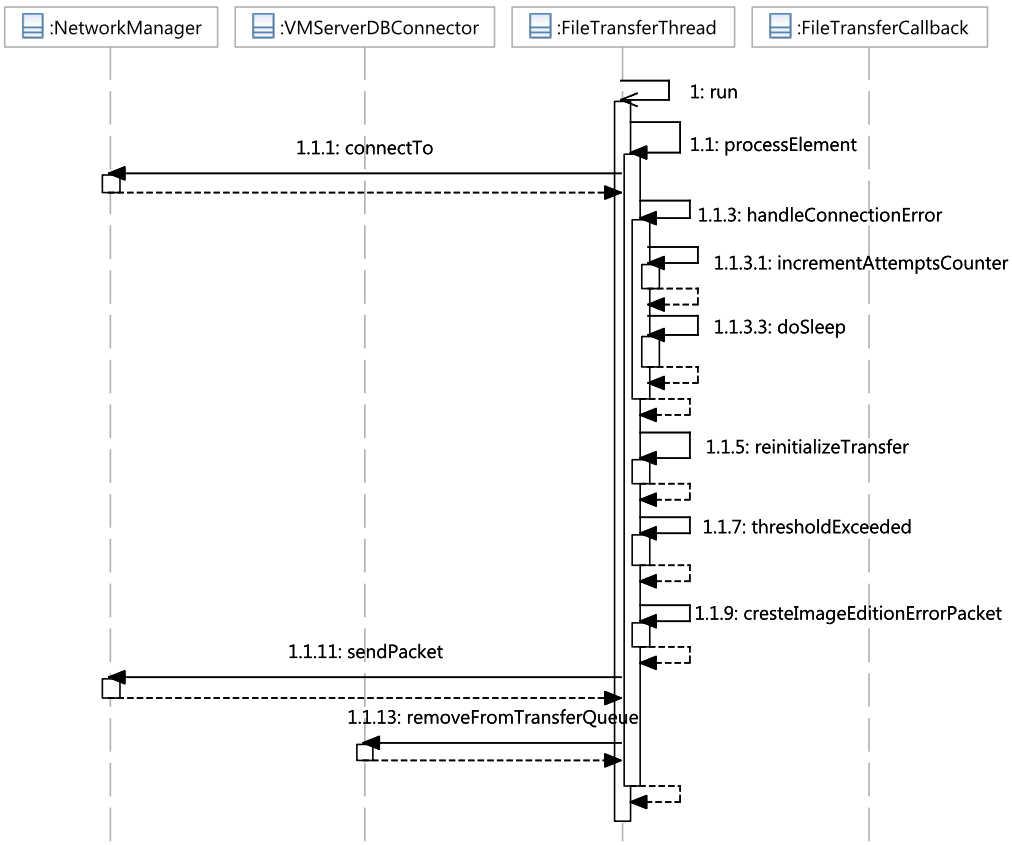

FIGURA 4.56: Creación y edición de imágenes de disco: error al establecer la conexión con repositorio de imágenes

Cuando el establecimiento de la conexión falla, se lanza una excepción. Al procesarla, el hilo de transferencias

1. actualiza un contador de intentos fallidos,
2. duerme durante $4^{\text{intentos fallidos}}$ segundos, y
3. reinicia el procesamiento de la transferencia
4. si el número de intentos de conexión fallidos supera un determinado umbral, que puede fijarse en el fichero de configuración del demonio del servidor de máquinas virtuales,

   a) se construye el paquete del tipo Error de edición de imagen,

   b) se envía dicho paquete al servidor de *cluster*, y

   c) se elimina la petición de la cola de transferencias.

Por otro lado, el diagrama de secuencia de la figura 4.57 muestra cómo se procesan los errores en las transferencias FTP. Nuevamente, los pasos iniciales del procesamiento de la petición siguen siendo iguales que en la secuencia básica, por lo que los hemos omitido. Además, la forma de proceder es idéntica a la del caso anterior.

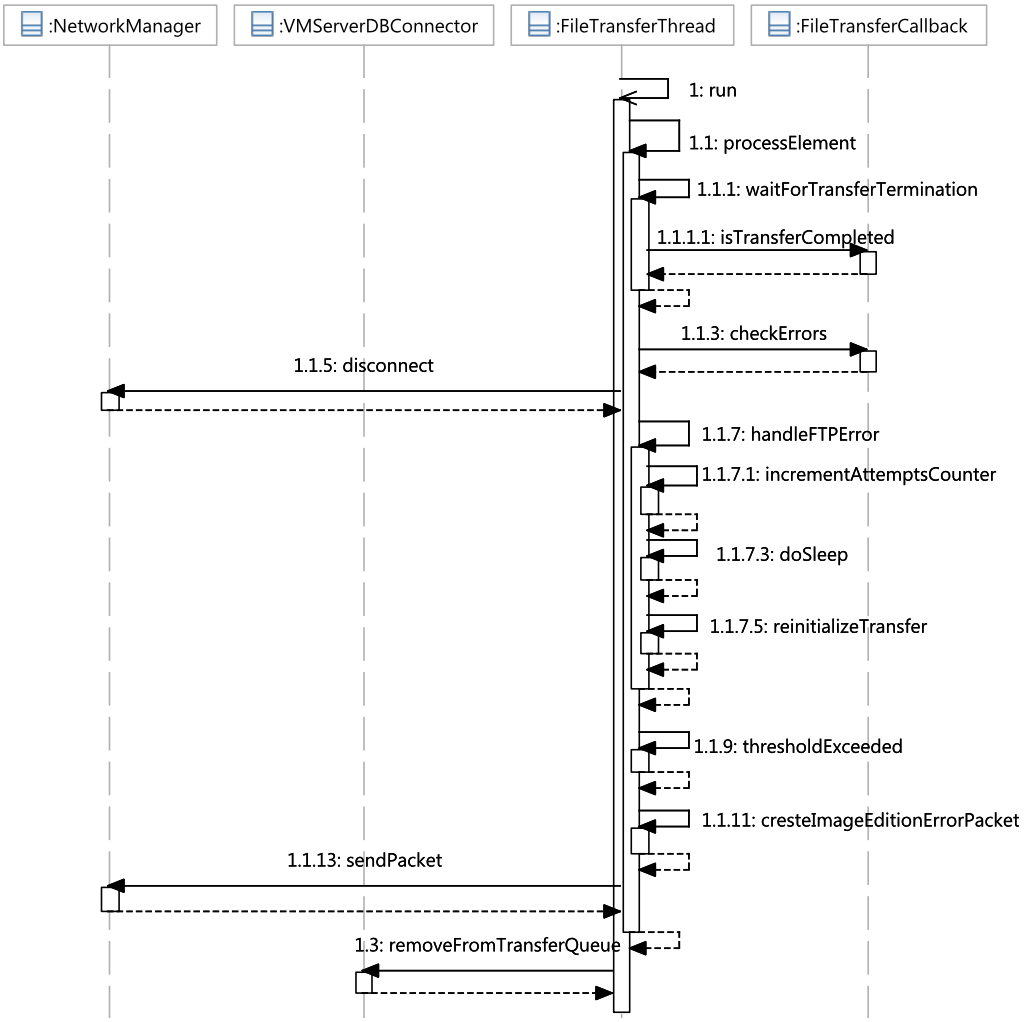

FIGURA 4.57: Creación y edición de imágenes de disco: error al realizar la transferencia FTP

Finalmente, siempre que falla el establecimiento de la conexión o una transferencia FTP, se utiliza el mismo contador de intentos fallidos.

4.5.6.9.3. Extracción del fichero comprimido

Una vez se ha descargado el fichero comprimido desde el repositorio de imágenes, es necesario extraer su contenido. Por convenio, el fichero comprimido contiene tres ficheros ubicados en su directorio raíz:

- el fichero `Definition.xml`, que es la plantilla que se usará para generar el fichero de configuración de todas las máquinas virtuales que usen la misma imagen.
- el fichero `OS.qcow2`, que es la imagen de disco que contiene el sistema operativo y los programas instalados. Durante el proceso de arranque de las máquinas virtuales, se crearán imágenes *copy-on-write* a partir de este fichero.
- el fichero `Data.qcow2`, que es la imagen de disco que contiene el fichero de paginación y los datos temporales que usan los usuarios. Durante el proceso de arranque de las máquinas virtuales, se creará una copia de este fichero.

Tras finalizar la extracción del fichero comprimido,

- los ficheros `.qcow2` se moverán al directorio de imágenes de disco, y
- el fichero `.xml` se moverá al directorio de ficheros de definición.

La ubicación de ambos se especifica en el fichero de configuración del demonio del servidor de máquinas virtuales.
Por otra parte, tras mover estos ficheros se creará la máquina virtual que se utilizará para editar las imágenes de disco. Todos estos pasos se reflejan en el diagrama de secuencia de la figura 4.58.
Es importante notar que

- siempre se comprueba que el fichero comprimido contiene los ficheros `OS.qcow2`, `Data.qcow2` y `Definition.xml`. En caso de que alguna de las imágenes de disco esté corrupta, el error se detectará al arrancar la máquina virtual.
- antes de utilizar las imágenes de disco en el arranque de la máquina virtual, se modifican sus permisos mediante la orden `chmod`. Así, nos aseguramos de que pueden leerse y escribirse en todos los casos.
- la petición se elimina de la cola cuando se ha procesado completamente. Por ello, si el demonio del servidor de máquinas virtuales termina de forma inesperada, el proceso de descompresión podrá reanudarse en su próximo arranque sin que sea necesario realizar nuevamente la transferencia con el repositorio de imágenes. Esto permite ahorrar tiempo y ancho de banda.
- los datos de conexión al repositorio de imágenes se almacenan en una tabla de la base de datos. Esto es necesario para transferir la nueva imagen o la imagen editada al repositorio de imágenes.

4.5.6.9.4. Errores en la extracción del fichero comprimido

Al extraer el fichero comprimido, pueden detectarse los siguientes errores:

- el fichero comprimido está corrupto. En estos casos, no se puede extraer.
- el contenido del fichero comprimido es incorrecto. En estos casos, falta alguno de los tres ficheros de los que hablamos en la sección anterior, o alguno de ellos contiene datos no válidos.
- falla la llamada de `libvirt` que crea la máquina virtual.

Todos estos errores se tratan de forma idéntica, por lo que nos centraremos en uno cualquiera de ellos: el segundo. El diagrama de secuencia de la figura 4.59 muestra cómo se realiza su tratamiento. Por claridad, en ese diagrama hemos omitido las fases previas a la detección del error, que son iguales que en la secuencia básica.
Como podemos observar en el diagrama de secuencia, para tratar el error se siguen los siguientes pasos:

1. se construye un paquete del tipo Error de edición de imagen.
2. se envía dicho paquete al servidor de *cluster*.
3. se crea y encola una transferencia especial. Durante su procesamiento, no se transferirán datos: sólo se interactuará con el repositorio de imágenes de la siguiente manera:
   - si se está creando una imagen, se liberará el identificador pedido,
   - si se está editando una imagen, se liberará el fichero comprimido en el repositorio de imágenes, y

   Ambas interacciones se realizan de acuerdo a lo que dijimos en la sección 4.5.5.8.5.

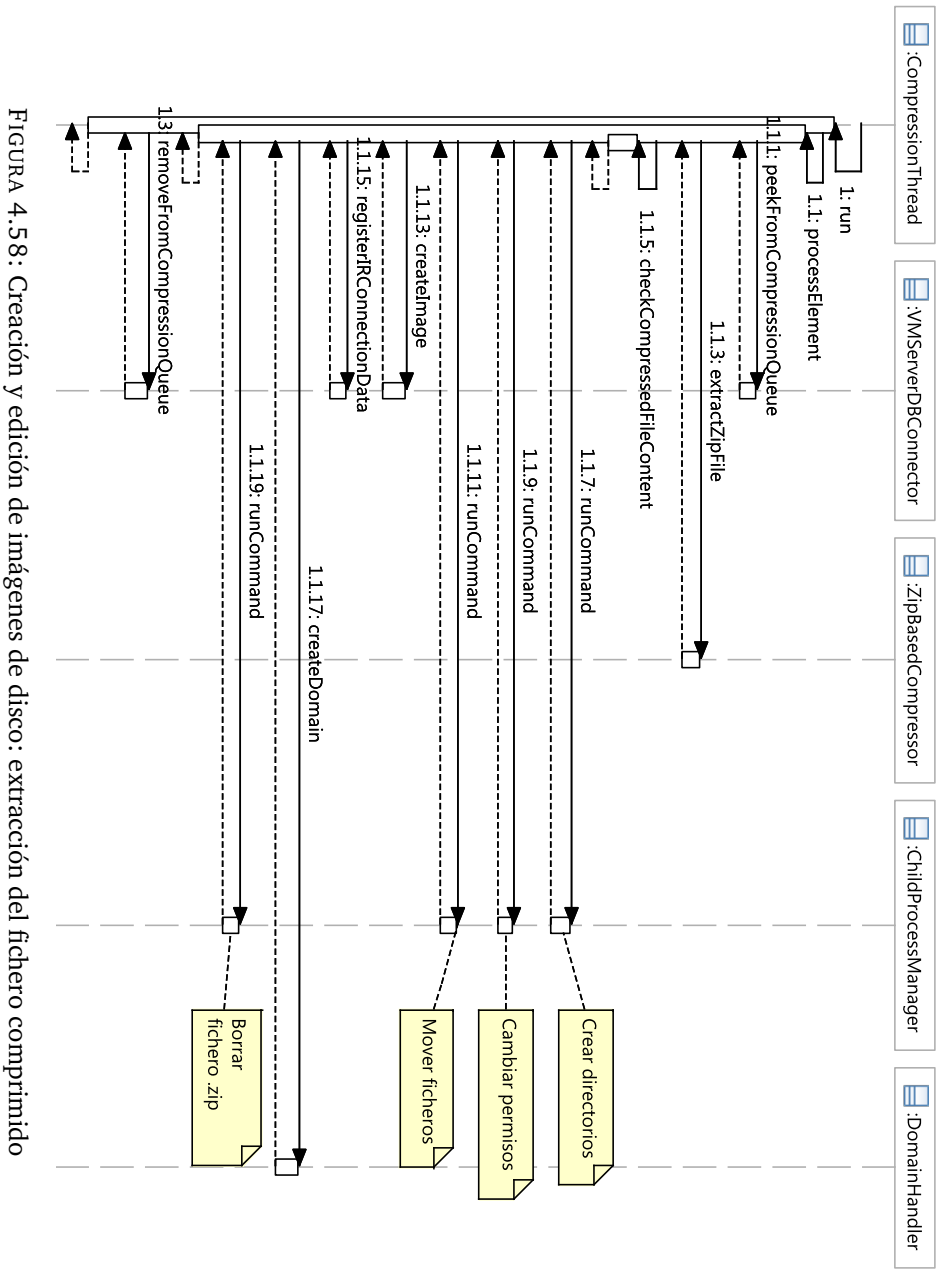

FIGURA 4.58: Creación y edición de imágenes de disco: extracción del fichero comprimido

4. se borran el fichero `.zip` y los ficheros extraídos (si existen).

Es importante notar que, cuando falla la llamada a `libvirt` que crea la máquina virtual, el objeto `DomainHandler` se ocupa de liberar los recursos asociados a la máquina virtual.

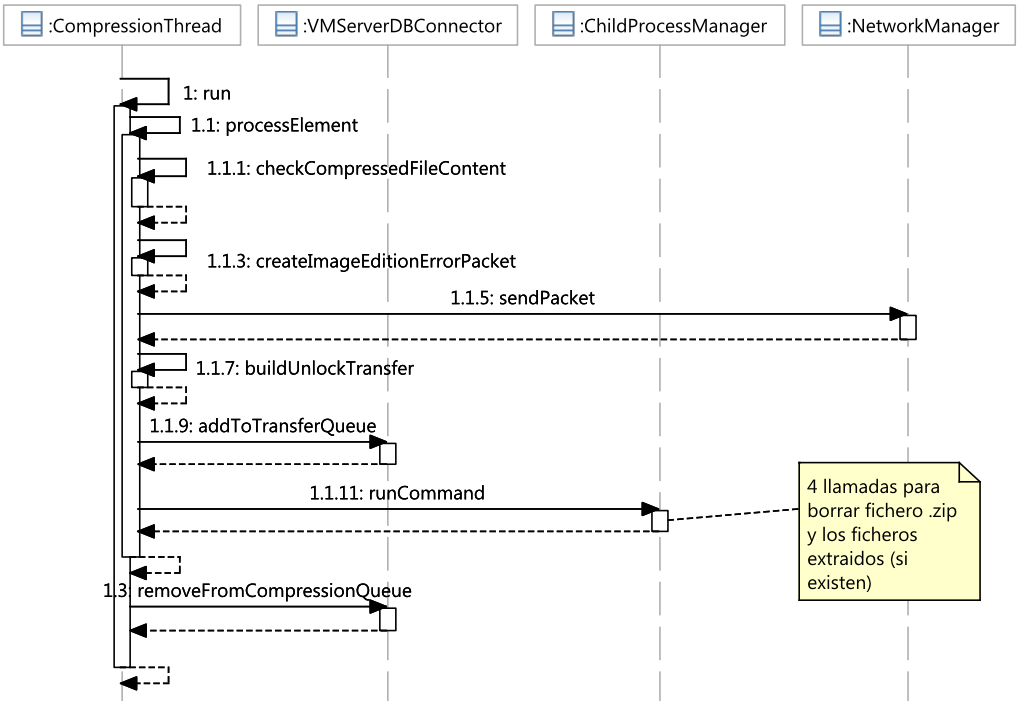

FIGURA 4.59: Creación y edición de imágenes de disco: fallo durante la extracción del fichero comprimido

4.5.6.9.5. Arranque de la máquina virtual

Cuando se arranca una máquina virtual que se utilizará para editar imágenes de disco, se realizan prácticamente las mismas acciones que vimos en las secciones 4.5.6.8.2 y 4.5.7.7.3.

Sólo es necesario introducir una pequeña modificación: no se creará la imagen *copy-on-write* ni una copia de la imagen que alberga los datos y el fichero de paginación.

Por otra parte, cuando se produce un error se procede como vimos en la sección 4.5.7.7.3. Por ello, no reproduciremos nuevamente las interacciones que presentamos en dicha sección.

4.5.6.9.6. Apagado de la máquina virtual

El proceso de apagado de una máquina virtual que se utiliza para crear o editar imágenes es muy similar al que presentamos en la sección 4.5.6.9.6. Nuevamente, en esta sección mostraremos las modificaciones que hay que realizar en él. El proceso de apagado modificado se recoge en el diagrama de secuencia de la figura 4.60.

Como podemos observar en dicho diagrama,

- no se borran las imágenes de disco, ya que es necesario añadirlas a un fichero comprimido, y
- se crea y encola una petición de compresión, que contiene todo lo necesario para generar el fichero comprimido.

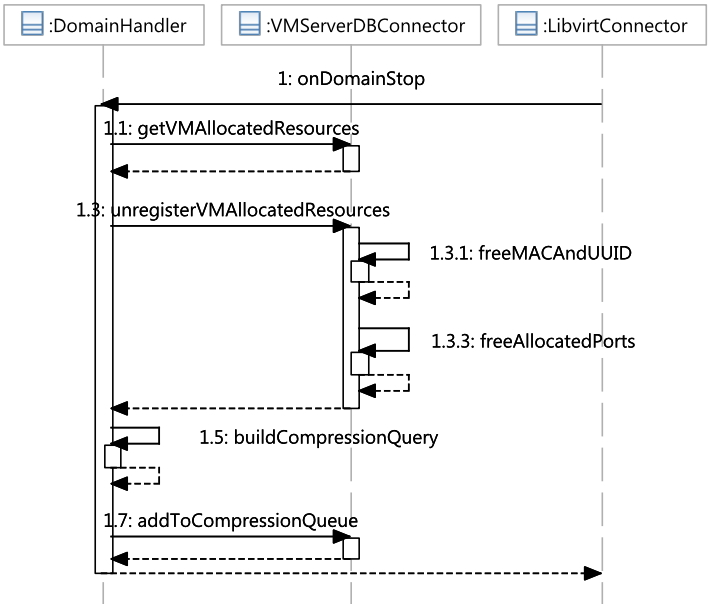

FIGURA 4.60: Creación y edición de imágenes: apagado de la máquina virtual

4.5.6.9.7. Creación del fichero comprimido

Una vez apagada la máquina virtual que se utiliza para crear o editar una imagen, se crea y encola una petición de compresión, que se procesará, en ausencia de errores, como se indica en el diagrama de secuencia de la figura 4.61.

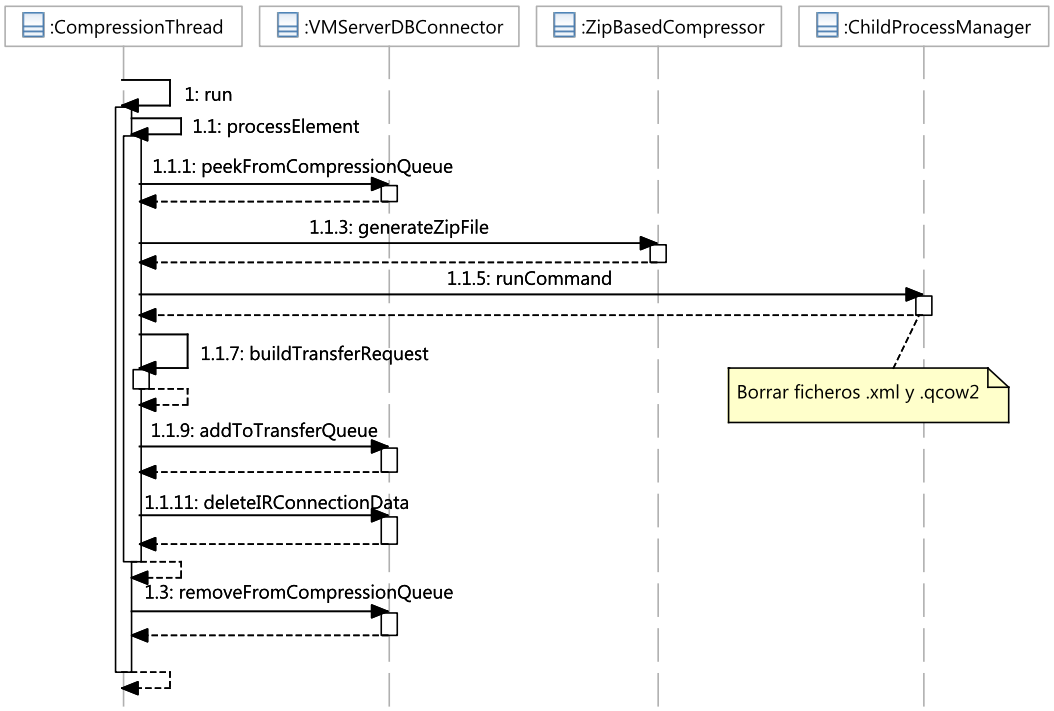

FIGURA 4.61: Creación y edición de imágenes: creación del fichero comprimido

Como podemos observar en el diagrama, se siguen los siguientes pasos:

1. se utiliza un objeto `ZipBasedCompressor` para generar el fichero comprimido.
2. se borran las imágenes de disco `.qcow2` y el fichero `.xml` de definición de la máquina virtual.
3. se construye una petición de transferencia y se añade a la cola de transferencias.
4. se borran los datos de conexión del repositorio de imágenes de la base de datos del servidor de máquinas virtuales.

Es importante notar que la petición se elimina de la cola cuando se ha procesado completamente. Por ello, si el demonio del servidor de máquinas virtuales termina de forma inesperada, este proceso podrá reanudarse en su próximo arranque sin que se pierdan datos.

Finalmente, en caso de que el proceso de compresión falle, se procederá exactamente igual que en la sección 4.5.6.9.4. Por ello, no reproduciremos nuevamente dicho proceso.

4.5.6.9.8. Subida del fichero comprimido

Tras generar el fichero comprimido, se generará y encolará una petición de transferencia. Siempre que no se produzcan errores, esta petición se procesará de acuerdo al diagrama de secuencia de la figura 4.62.

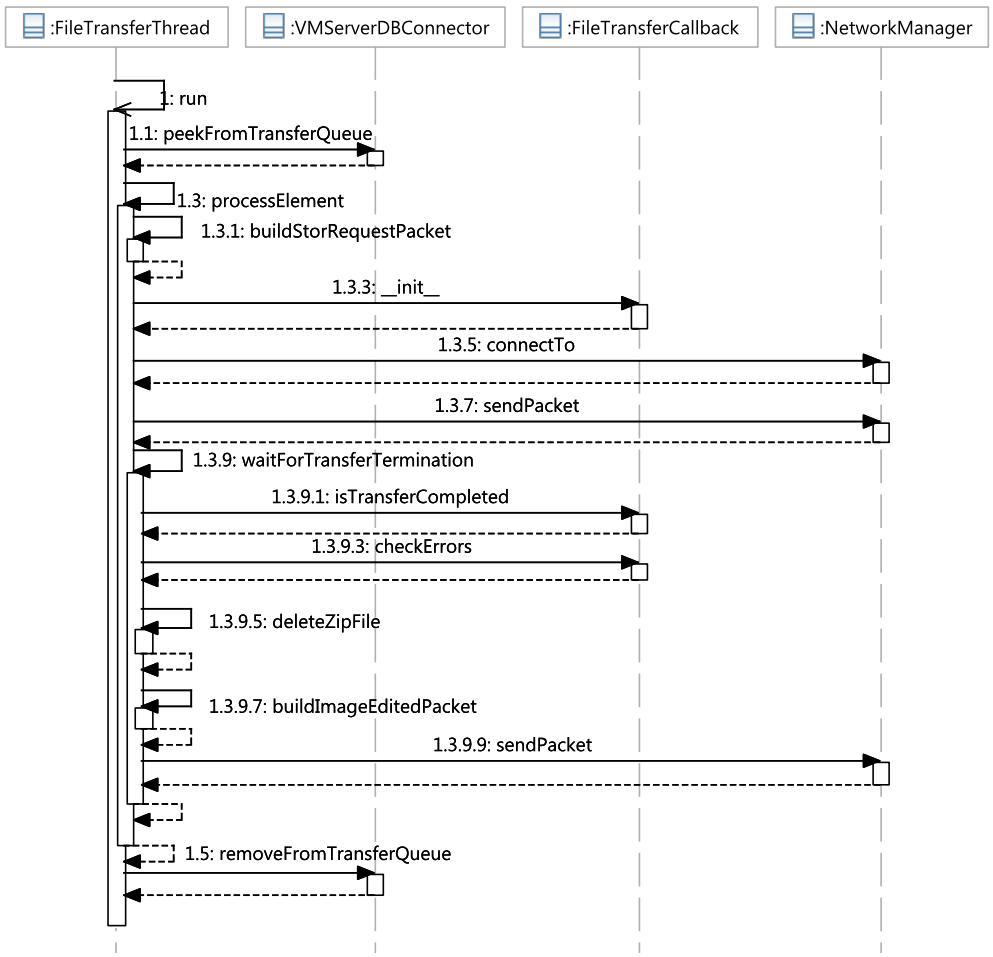

FIGURA 4.62: Creación y edición de imágenes: subida del fichero comprimido al repositorio de imágenes

Como podemos observar en dicho diagrama, se siguen los siguientes pasos:

1. se crea un paquete del tipo Petición `FTP STOR`.
2. se establece la conexión con el repositorio de imágenes y se le envía dicho paquete.
3. cuando la transferencia se completa, se comprueba si se ha producido algún error. En este caso, no se ha producido ninguno.
4. se borra el fichero `.zip`.
5. se construye un paquete del tipo Imagen editada. Este contiene, además del identificador único de la petición, el identificador único de la imagen que ha terminado de editarse.
6. se envía dicho paquete al servidor de *cluster*.

Nuevamente, la petición de transferencia sólo se elimina de la cola tras realizar todas estas acciones. Así, si se interrumpe este proceso, se reanudará en el siguiente arranque del servidor de máquinas virtuales, lo que impide que se pierdan datos.

Asimismo, en caso de que se produzca algún error, se procederá de acuerdo a lo indicado en la sección 4.5.6.9.2. Por ello, no volveremos a reproducir aquí esas interacciones.

4.5.6.10. Despliegue de imágenes de disco

El proceso de despliegue de imágenes de disco comienza cuando el servidor de *cluster* envía al servidor de máquinas virtuales un paquete del tipo Desplegar imagen. Dicho paquete contiene, además del identificador único de la petición, el identificador único de la imagen a desplegar.

Durante la ejecución de este proceso, podemos distinguir dos fases: la descarga del fichero comprimido desde el repositorio de imágenes y la descompresión y el procesamiento del contenido de ficho fichero.

La fase de descarga del fichero comprimido es idéntica a la que tiene lugar durante los procesos de creación y edición de imágenes de disco. Puesto que hablamos detalladamente de esta fase en las secciones 4.5.6.9.1 y 4.5.6.9.2, no volveremos a reproducirla aquí.

Por otra parte, la fase de descompresión y procesamiento del contenido del fichero comprimido es también muy similar a la que tiene lugar durante los procesos de creación y edición de imágenes de disco, que ya estudiamos en las secciones 4.5.6.9.3 y 4.5.6.9.4. Por ello, nos limitaremos a mostrar las diferencias, que son las siguientes:

- tras colocar las imágenes de disco en la ubicación correcta, no se arranca ninguna máquina virtual.
- no es necesario almacenar los datos de conexión al repositorio de imágenes en la base de datos del servidor de máquinas virtuales. Esto se debe a que, una vez finalizada la descarga del fichero comprimido, para atender la petición de despliegue no es necesario volver a interactuar con el repositorio de imágenes.
- tras finalizar la descompresión, se crea y envía un paquete del tipo Imagen desplegada al servidor de *cluster*. Dicho paquete contiene el identificador único de la petición y el identificador de la imagen desplegada.

Finalmente, si se produce algún error durante esta fase, se procederá como vimos en la sección 4.5.6.9.4. No obstante, en este caso los paquetes de error serán del tipo Error de despliegue, y no del tipo Error de edición de imagen. Ambos tipos de paquete tienen el mismo contenido.

4.5.6.11. Borrado de imágenes de disco

Este proceso hace que un servidor de máquinas virtuales deje de albergar una imagen.

Como viene siendo habitual, el proceso de borrado de una imagen se inicia cuando el servidor de *cluster* envía un paquete del tipo Borrar imagen al servidor de máquinas virtuales. Estos paquetes contienen el identificador único de la petición y el identificador único de la imagen a borrar, y se procesan de acuerdo al diagrama de secuencia de la figura 4.63.

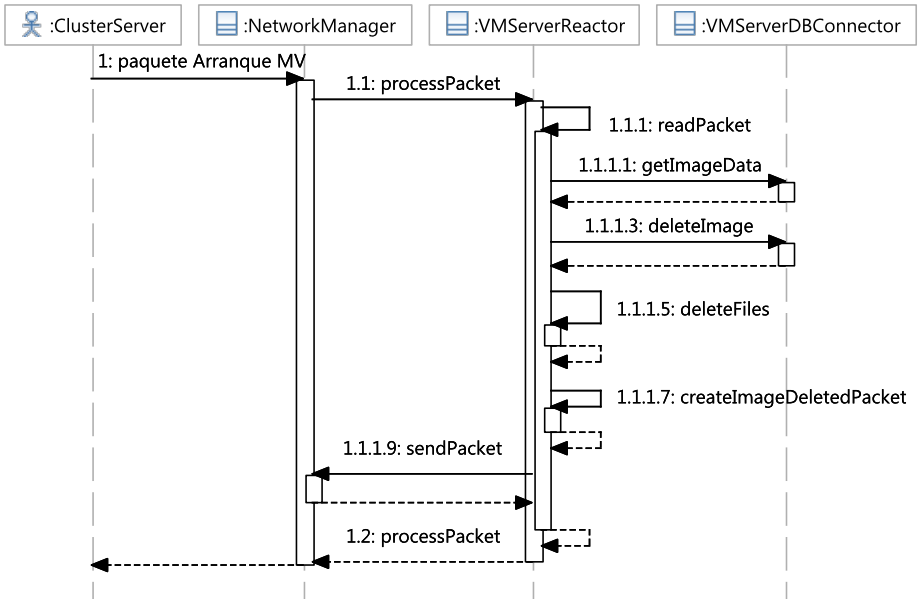

FIGURA 4.63: Borrado de imágenes de disco: flujo básico

Como podemos observar en el diagrama, se siguen los siguientes pasos:

1. tras leer el paquete, se extraen de la base de datos las rutas de los ficheros a borrar.
2. acto seguido, se borra toda la información asociada a la imagen de la base de datos del servidor.
3. se borran las imágenes de disco y el fichero de definición.
4. se construye un paquete del tipo Imagen borrada. Este contiene el identificador único de la petición y el identificador único de la imagen que se ha borrado del servidor.
5. se envía dicho paquete al servidor de *cluster*.

Es importante notar que, si alguna máquina virtual está utilizando las imágenes de disco a borrar, el sistema de ficheros del servidor de máquinas virtuales nos garantizará que no se producirán errores: las imágenes de disco sólo se borrarán cuando ninguna máquina virtual los utilice.

Por otra parte, en caso de que

- el servidor de máquinas virtuales no disponga de las imágenes de disco a borrar, que
- se produzca un error al eliminar los ficheros, o que
- se produzca un error al realizar la actualización de la base de datos

se abortará el proceso de borrado y se procederá de la siguiente manera:

1. se construye el paquete del tipo Error de borrado de imagen. Estos paquetes tienen la misma información que los paquetes del tipo Imagen borrada.
2. se envía dicho paquete al servidor de *cluster*.

4.5.6.12. Apagado forzoso de una máquina virtual

Como vimos en la sección 4.5.6.8.7, los usuarios pueden provocar el reinicio forzoso de las máquinas virtuales. Sin embargo, en algunos casos los problemas pueden no resolverse de este modo. Por ello, también es posible realizar el apagado forzoso de una máquina virtual.

Para desencadenarlo, el servidor de *cluster* enviará un paquete del tipo Destruir dominio al servidor de máquinas virtuales. Este contiene el identificador único de la máquina virtual a apagar.

El diagrama de secuencia de la figura 4.64 muestra cómo se procesa una de estas peticiones

Los pasos que se siguen son muy similares a los del reinicio forzoso de una máquina virtual, que estudiamos en la sección 4.5.6.8.7. No obstante, en este caso se hace una llamada a `libvirt` para destruir el dominio.

Cuando la máquina virtual se apague, `libvirt` generará un evento de apagado que que procesará como hemos visto en las secciones 4.5.6.9.6 y 4.5.6.9.6.

4.5.6.13. Directorios del servidor de máquinas virtuales

El demonio del servidor de máquinas virtuales utiliza cuatro directorios:

- el **directorio de ficheros de definición**, en el que se almacenan los ficheros de definición asociados a las imágenes que amacena el servidor.
- el **directorio de imágenes de disco**, en el que se almacenarán las imágenes de disco .qcow2 que se usarán para arrancar máquinas virtuales.
- el **directorio de imágenes en uso**, en el que se almacenarán las imágenes de disco utilizadas por las máquinas virtuales activas, es decir, las imágenes *copy-on-write* y las imágenes de disco con los ficheros de paginación y los datos temporales.
- el **directorio de transferencias**, en el que se almacenarán y extraerán los ficheros `.zip` que se intercambian con el repositorio de imágenes.

La ubicación de todos estos directorios se especifica en el fichero de configuración del demonio del servidor de máquinas virtuales.

El uso de varios directorios permite realizar optimizaciones que mejoran el rendimiento de las máquinas virtuales. Por ejemplo, si aprovechamos el hecho de que, salvo al desplegar, crear o editar imágenes, nunca se escribe en el directorio de imágenes de disco, podemos mejorar notablemente el rendimiento de las máquinas virtuales si

- el directorio de imágenes de disco está ubicado en una unidad SSD (*Solid State Drive*, unidad de estado sólido), y
- los directorios restantes están ubicados en un disco duro convencional.

4.5.6.14. Formatos de paquete

En esta sección, mostraremos detalladamente las características de los tipos de paquete asociados al subsistema servidor de máquinas virtuales. El cuadro 4.7 muestra los códigos, prioridades y las constantes del tipo enumerado `PACKET_T` (definido en el paquete `virtualMachineServer.packetHandling`) asociados a cada clase de paquete.

Como dijimos en la sección 4.5.3.3, cuanto menor es la valor numérico de la prioridad de un paquete, más prioritario es.

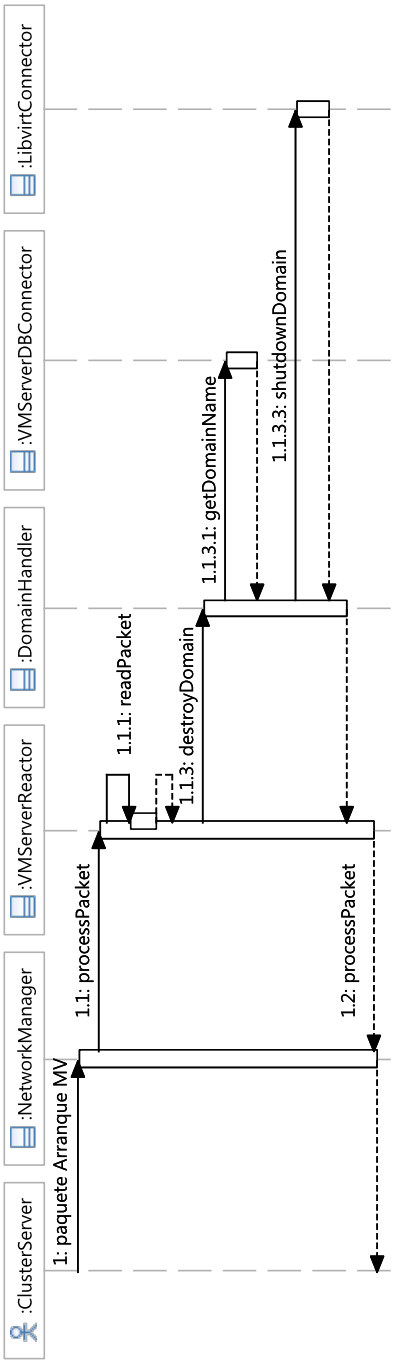

FIGURA 4.64: Apagado forzoso de una máquina virtual

Tipo de paquete	Código	Prioridad	Constante (tipo enumerado)
Creación MV	1	5	CREATE_DOMAIN
Apagado forzoso MV	2	5	DESTROY_DOMAIN
Reinicio forzoso MV	3	5	REBOOT_DOMAIN
Datos de conexión de MV	4	5	DOMAIN_CONNECTION_DATA
Estado del servidor MVs	5	5	SERVER_STATUS
Solicitud de estado	6	5	SERVER_STATUS_REQUEST
Apagado con espera	7	3	USER_FRIENDLY_SHUTDOWN
Apagado inmediato	8	3	HALT
Solicitud de datos de conexión VNC	9	5	QUERY_ACTIVE_VM_DATA
Datos de conexión VNC	10	5	ACTIVE_VM_DATA
Solicitud IDs MVs	11	5	QUERY_ACTIVE_DOMAIN_UIDS
IDs de MVs	12	5	ACTIVE_DOMAIN_UIDS
Editar imagen	13	5	IMAGE_EDITION
Error de edición de imagen	14	4	IMAGE_EDITION_ERROR
Desplegar imagen	15	5	DEPLOY_IMAGE
Error de despliegue	16	4	IMAGE_DEPLOYMENT_ERROR
Borrar imagen	17	5	DELETE_IMAGE
Error de borrado	18	4	IMAGE_DELETION_ERROR
Imagen editada	19	5	IMAGE_EDITED
Imagen desplegada	20	5	IMAGE_DEPLOYED
Imagen borrada	21	5	IMAGE_DELETED
Error interno	22	5	INTERNAL_ERROR

CUADRO 4.7: Características principales de los paquetes utilizados en el servidor de máquinas virtuales

El contenido de los distintos tipos de paquete es el siguiente:

- los paquetes del tipo **Creación de máquina virtual** contienen el identificador único de la petición, el identificador del usuario que la ha realizado y el identificador único asociado a las imágenes de disco de la máquina virtual a arrancar. Los dos últimos son valores enteros, y el restante es un *string*.

- los paquetes de los tipos **Apagado forzoso de máquina virtual** y **Reinicio forzoso de máquina virtual** contienen el identificador único de la máquina virtual a apagar o a reiniciar. Dicho identificador es un *string*.

- los paquetes del tipo **Datos de conexión a máquina virtual** contienen el identificador único de la petición, la dirección IP, el puerto y la contraseña del servidor VNC asociado a una máquina virtual. Salvo el puerto, que es un valor entero, el resto de datos del paquete son *strings*.

- los paquetes del tipo **Estado del servidor de máquinas virtuales** contienen la dirección IP del servidor de máquinas virtuales (un *string*) y valores enteros que indican el número de máquinas virtuales activas, las cantidades de memoria RAM ocupada y libre y el espacio utilizado y disponible en el directorio de almacenamiento de imágenes y en el directorio de datos temporales.

- los paquetes de los tipos **Solicitud de estado**, **Solicitud de identificadores de máquinas virtuales**, **Solicitud de datos de conexión VNC** y **Apagado inmediato** no contienen información adicional.

- los paquetes del tipo **Apagado con espera** contienen un valor entero que indica el tiempo de espera en segundos.

- los paquetes del tipo **Datos de conexión VNC** contienen la dirección IP del servidor de máquinas virtuales (un *string*), así como los puertos y las contraseñas de todos los servidores VNC asociados a las máquinas virtuales que alberga el servidor de máquinas virtuales.
- los paquetes del tipo **Identificadores de máquinas virtuales** contienen la dirección IP del servidor de máquinas virtuales y los identificadores de todas las máquinas virtuales que alberga, que son *strings*.
- los paquetes del tipo **Editar imagen** contienen
  - el identificador único de la petición, que es un *string*.
  - el identificador único del usuario que la ha generado.
  - la dirección IP (un *string*) y el puerto (un valor entero) del repositorio de imágenes.
  - el identificador único de las imágenes de disco a descargar del repositorio de imágenes (un valor entero).
  - un *flag* (valor entero) que indica si las imágenes de disco que se descargarán del repositorio de imágenes se van a editar o a utilizar como base para crear otras máquinas virtuales.
- los paquetes de los tipos **Error de edición de imagen**, **Error de despliegue** y **Error de borrado** contienen el identificador único de la petición (un *string*) y el identificador único de las imágenes de disco a las que esta afecta (un valor entero).
- los paquetes del tipo **Desplegar imagen** contienen
  - el identificador único de la petición, que es un *string*.
  - la dirección IP (un *string*) y el puerto (un valor entero) del repositorio de imágenes.
  - el identificador único de las imágenes de disco a descargar del repositorio de imágenes (un valor entero).
- los paquetes del tipo **Borrar imagen** contienen el identificador único de la petición (un *string*) y el identificador único de las imágenes de disco a borrar del servidor de máquinas virtuales (un valor entero).
- finalmente,los paquetes de los tipos **Imagen editada**, **Imagen desplegada**, **Imagen borrada** y **Error interno** contienen el identificador único de la petición (un *string*)

4.5.6.15. Esquema de la base de datos

La base de datos del servidor de máquinas virtuales contiene las siguientes tablas:

- `Image`. Esta tabla almacena las rutas relativas de los ficheros de definición y de las distintas imágenes de disco almacenadas en el servidor de máquinas virtuales. Sus columnas son las siguientes:
  - `imageID`. Es el identificador asociado a un par de imágenes de disco, y la clave primaria de esta tabla. Se almacena como un valor entero.
  - `osImagePath`, `dataImagePath`, `definitionFilePath`: son las rutas relativas de la imagen con el sistema operativo y los programas, la imagen que almacenará los datos temporales y el fichero de paginación y el fichero de definición que se usarán para crear una máquina virtual. Estos valores son cadenas de caracteres de hasta 100 *bytes* de longitud.
  - `bootable`: se trata de un bit que indica si se está editando el contenido de las imágenes de disco o no.
- `ActiveVM`. Esta tabla almacena los recursos asociados a las máquinas virtuales activas. Incluye las siguientes columnas:

  - `domainName`. Es la clave primaria de la tabla. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud que contiene el nombre de la máquina virtual que, como sabemos, debe ser único dentro del servidor de máquinas virtuales.
  - `imageID`. Como ya hemos visto, este valor entero es el identificador único asociado a las imágenes de disco que usa la máquina virtual.
  - `VNCPort`. Se trata de un valor entero, que se corresponde con el puerto par reservado para el servidor VNC de la máquina virtual. Como dijimos en la sección 4.5.6.8.2, el *websocket* tiene asociado un puerto impar: el puerto consecutivo al del servidor VNC. Por ello, no es necesario almacenar este valor en la base de datos.
  - `VNCPass`. Se trata de una secuencia de caracteres de 65 *bytes* de longitud, que contiene la contraseña del servidor VNC.
  - `userID`. Se trata de un valor entero, que se corresponde con el identificador único del usuario que envió la petición de arranque de máquina virtual o de creación o edición de imágenes de disco.
  - `osImagePath`, `dataImagePath`. Se trata de dos cadenas de caracteres de 100 *bytes* de longitud, que contienen las rutas absolutas de las dos imágenes de disco que usa la máquina virtual.
  - `websockifyPID`. Se trata de un valor entero, que almacena el PID del proceso *websockify*. Cuando este no se arranca, toma el valor 0.
  - `macAddress`. Se trata de una cadena de caracteres de 20 *bytes* de longitud, que contiene la dirección MAC de la máquina virtual.
  - `uuid`. Se trata de una cadena de caracteres de 40 *bytes*, que contiene el UUID de la máquina virtual.

- `ActiveDomainUUIDs`. Esta tabla asocia a cada nombre de máquina virtual un identificador único, que identifica a la máquina virtual en toda la infraestructura. Tiene dos columnas:

  - `domainName`. Es la clave primaria de la tabla. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud que contiene el nombre de la máquina virtual.
  - `commandID`. Se trata de una cadena de caracteres de hasta 70 *bytes*, que identifica unívocamente a la petición de arranque, creación o edición de imagen que creó la máquina virtual.

- `CompressionQueue`. Esta tabla almacena los elementos de la cola de compresión. Contiene las siguientes columnas:

  - `position`. Se trata de un valor entero, que es la clave primaria de la tabla.
  - `data`. Almacena cadenas de caracteres de hasta 420 *bytes* de longitud, que contienen las peticiones de la cola de compresión serializadas.

- `TransferQueue`. Esta tabla almacena los elementos de la cola de transferencias. Contiene las siguientes columnas:

  - `position`. Se trata de un valor entero, que es la clave primaria de la tabla.
  - `data`. Almacena cadenas de caracteres de hasta 230 *bytes* de longitud, que contienen las peticiones de la cola de transferencias serializadas.

- `ConnectionDataDictionary`. Esta tabla almacena los datos de conexión al repositorio de imágenes, y contiene las siguientes columnas:

  - `dict_key`. Se trata de una cadena de caracteres de hasta 70 *bytes* de longitud. Almacenará el identificador único de una petición de creación o edición de imágenes de disco.

    - `value`. Se trata de una cadena de caracteres de hasta 21 *bytes* de longitud que contiene la dirección IP y el puerto del repositorio de imágenes asociado a la petición. Esta información está serializada.

- `FreeMACsAndUUIDs`. Esta tabla contiene las siguientes columnas:
    - `mac`. Se trata de una cadena de caracteres de 20 *bytes* de longitud, que contiene una dirección MAC. Es la clave primaria de la tabla.
    - `uuid`. Se trata de una cadena de caracteres de 40 *bytes*, que contiene un UUID.
- `FreeVNCPorts`. Esta tabla contiene una única columna, `vncPort`. Se trata de un valor entero que se corresponde con un puerto y que, por tanto, puede tomar valores entre 1 y 65535. Asimismo, esta columna es la clave primaria de esta tabla.

Finalmente, las tablas `FreeMACSAndUUIDs` y `FreeVNCPorts` residen en memoria. El resto de tablas de la base de datos residen en disco.

4.5.7. El paquete `clusterServer`

Como dijimos en la sección 4.4, las principales funciones de los servidores de *cluster* son las siguientes:

- realizar el balanceado de carga entre los distintos servidores de máquinas virtuales del *cluster*.
- arrancar, detener, dar de alta y borrar servidores de máquinas virtuales.
- atender todas las peticiones de los usuarios. Para ello, debe
    - comprobar que el *cluster* es capaz de atenderlas, informando de errores cuando sea preciso.
    - detectar los errores que se producen al procesarlas, informando a los usuarios cuando sea preciso.
    - recopilar periódicamente el estado del resto de máquinas del *cluster*.

En esta sección, presentaremos en detalle su diseño. Por claridad, y salvo indicación en contra, de ahora en adelante utilizaremos el término servidor de *cluster* para referiremos al demonio del servidor de *cluster*. Este proceso reside en el servidor que actúa como servidor de *cluster*.

4.5.7.1. Balanceado de carga

El balanceado de carga es una de las funciones más importantes del servidor de *cluster*. Puesto que el rendimiento de la infraestructura se ve muy afectado por el algoritmo de balanceado de carga que se utilice, hemos decidido simplificar, en la medida de lo posible, el cambio del algoritmo de balanceado de carga utilizado. Para ello, hemos definido la interfaz `LoadBalancer`, común a todos los algoritmos de balanceado de carga.

Asimismo, hemos implementado un algoritmo de carga parametrizable que garantiza que la carga de trabajo de todos los servidores de máquinas virtuales sea muy similar. La clase `PenaltyBasedLoadBalancer` implementa dicho algoritmo.

Por claridad, empezaremos mostrando el algoritmo de balanceado de carga. Acto seguido, mostraremos las principales relaciones de las clases `LoadBalancer` y `PenaltyBasedLoadBalancer`.

4.5.7.1.1. Descripción del algoritmo

La figura 4.65 muestra una implementación en *pseudocódigo* del algoritmo de balanceado de carga basado en penalizaciones.

Como podemos observar en esa figura, se siguen los siguientes pasos:

1. en función de la operación a realizar, se obtiene el conjunto de todos los servidores de máquinas virtuales *arrancados* que, *a priori*, pueden considerarse.

2. se comprueba que el servidor de máquinas virtuales puede utilizarse para realizar la operación. Si no es posible (porque, por ejemplo, no tiene suficiente memoria RAM libre), el servidor de máquinas virtuales se descartará.

   Por claridad, hemos omitido en el pseudocódigo los cálculos de las penalizaciones por el número de CPUs, la memoria RAM libre, el espacio de almacenamiento y el espacio de almacenamiento temporal. Dichos cálculos se hacen de acuerdo a las siguientes fórmulas:

$$\text{penalización CPUs} = \begin{cases} \dfrac{\text{CPUs}_i + \text{CPUs}_v}{\text{CPUs}_p} & \text{si modo} \neq \texttt{DESPLEGAR IMAGEN} \\ \dfrac{\text{CPUs}_i}{\text{CPUs}_p} & \text{en otro caso} \end{cases}$$

$$\text{penalización RAM} = \begin{cases} \dfrac{\text{RAM}_i + \text{RAM}_u}{\text{RAM}_t} & \text{si modo} \neq \texttt{DESPLEGAR IMAGEN} \\ \dfrac{\text{RAM}_i}{\text{RAM}_t} & \text{en otro caso} \end{cases}$$

$$\text{penalización almacenamiento} = \frac{\text{OS}_i - \text{Alm}_u}{\text{Alm}_t}$$

$$\text{penalización almacenamiento temporal} = \frac{\text{Datos}_i - \text{AlmTemporal}_u}{\text{AlmTemporal}_t}$$

   donde CPUs_i, RAM_i, OS_i y Datos_i son el número de CPUs, la cantidad de memoria RAM y los tamaños de las imágenes del sistema operativo y datos temporales de la máquina virtual.

   Asimismo, CPUs_v, CPUs_p, RAM_u, RAM_i, Alm_u, Alm_t, AlmTemporal_u y AlmTemporal_i son el número de CPUs virtuales y físicas, la cantidad de memoria RAM usada y total, el espacio de almacenamiento usado y total y el espacio de almacenamiento temporal usado y total en el servidor de máquinas virtuales.

   Es importante notar que el número de CPUs virtuales puede superar el número de CPUs físicas. Esto resulta interesante cuando las máquinas virtuales que se ubicarán en los servidores de máquinas virtuales utilizan muy poco la CPU. Por ejemplo, esto ocurrirá con mucha frecuencia si las máquinas virtuales sólo se utilizan para ejecutar *suites* ofimáticas.

   El exceso máximo admitido se puede configurar mediante el parámetro `exceso cpus` del algoritmo de balanceado de carga.

3. para cada uno de estos servidores, se calcula un factor de penalización. Estos están normalizados entre 0 y 1, y representan la medida en que se penalizará el rendimiento. Cuanto menores sean, menor se penalizará el rendimiento.

   Los factores de penalización se calculan de la siguiente manera:

$$\begin{aligned} \text{fp} = {} & p_{\text{CPUs}} \cdot \text{penalización CPUs} + p_{\text{RAM}} \cdot \text{penalización RAM} + \\ & p_{\text{Alm}} \cdot \text{penalización almacenamiento} + \\ & p_{\text{AlmTemp}} \cdot \text{penalización almacenamiento temporal} \end{aligned}$$

   siendo p_{CPUs}, p_{RAM}, p_{Alm} y p_{AlmTemp} los restantes parámetros de configuración del algoritmo de balanceado de carga.

```
fun algoritmoBCargaPenalizacion (idImagen : ent, modo : tmodo, copias_objetivo : ent,
      conector : conectorBD) dev
      <idsServidores : lista de ent, copias : ent, codigoError : ent>

      casos
            ◊ modo = ARRANQUE NORMAL:
                  servidores := conectorBD.obtenerServidores(idImagen)
            ◊ modo = CREAR O EDITAR IMAGEN:
                  servidores := conectorBD.obtenerServidoresEdicion(idImagen)
            ◊ modo = DESPLEGAR IMAGEN:
                  servidores := conectorBD.obtenerServidoresCandidatos(idImagen)
      fcasos

      caracImagen := conectorBD.obtenerCaractFamilia(idImagen)
      penalizaciones := lista_vacía()

      para cada idServidor en servidores hacer
            estado_servidor := conectorBD.obtenerEstado(idServidor)
            penalizacion_cpus := calcularPenalizacionCPUs()
            penalizacion_ram := calcularPenalizacionRAM()
            penalizacion_alm := calcularPenalizacionAlmacenamiento()
            penalizacion_alm_temporal :=
                  calcularPenalizacionAlmacenamientoTemporal()

            si (penalizacion_cpus ≤ 1 + exceso_cpus y penalizacion_ram ≤ 1
                  y penalizacion_alm ≤ 1 y penalizacion_alm_temporal ≤ 1) entonces
                  penalizacion_global := calcularPenalizacionGlobal()
                  casos
                        ◊ modo = DESPLEGAR IMAGEN:
                              copias_con_cpus := estadoServidor.numCPUs /
                                    caracImagen.cpus
                              copias_con_ram := estadoServidor.tamannoRAM /
                                    caracImagen.ram
                              copias_con_alm_temp :=
                                    estadoServidor.espacio_temp_disponible /
                                          caracImagen.espacio_temp
                              copias := mín(copias_con_cpus, copias_con_ram,
                                    copias_con_alm_temp
                              penalizaciones.annadir(crear_tupla(idServidor,
                                    penalizacion_global, copias))
                         ◊ modo != DESPLEGAR IMAGEN:
                              penalizaciones.annadir(crear_tupla(idServidor,
                                    penalizacion_global))
                  fasos
            fsi
      fpara

      si es_lista_vacía(penalizaciones) entonces
            devolver (lista_vacía, 0, codigo_error)
      si no
            ordenar_por_penalizacion_global(penalizaciones)
            resultado = lista_vacía ; copias := 0
            casos
                  ◊ modo = DESPLEGAR IMAGEN:
                        mientras copias < copias_objetivo :
                              añadir_a_lista(resultado,
                                    id_cabeza(penalizaciones)
                              copias += copias_cabeza(penalizaciones)
                              quitar_cabeza(penalizaciones)
                  ◊ modo != DESPLEGAR IMAGEN:
                        resultado = lista_unitaria(id_cabeza(penalizaciones))
            fcasos
      fsi
      devolver (resultado, copias, no_error)
ffun
```

FIGURA 4.65: Algoritmo de balanceado de carga basado en penalizaciones

4. si se va a realizar una operación de despliegue de imagen, se calculará el número de copias de la misma que pueden almacenarse en el servidor de máquinas virtuales.
5. se ordenan, de menor a mayor penalización, los servidores de máquinas virtuales que pueden utilizarse.
6. se escoge el servidor o servidores de máquinas virtuales con menor penalización.

Por tanto, este algoritmo de balanceado de carga distribuye las peticiones de forma que, siempre que sea posible, el rendimiento de todos los servidores de máquinas virtuales se degrade en la misma medida.

Finalmente, tal y como dijimos en la sección 4.4.20, las imágenes pueden crearse y editarse en ciertos servidores de máquinas virtuales reservados para ello. En caso de que la carga de trabajo del resto de servidores de máquinas virtuales sea muy elevada, estas máquinas también podrán utilizarse para desplegar imágenes de disco y, por tanto, para albergar máquinas virtuales de los usuarios.

4.5.7.1.2. La clase `PenaltyBasedLoadBalancer`

Como dijimos en la sección 4.65, la clase `PenaltyBasedLoadBalancer` implementa el algoritmo de balanceado de carga basado en penalizaciones. El diagrama de clases de la figura 4.66 contiene sus principales relaciones.

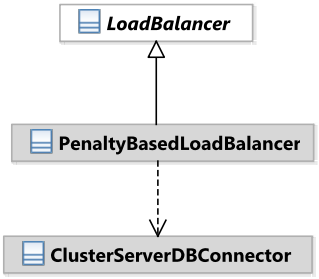

FIGURA 4.66: Principales relaciones de la clase `PenaltyBasedLoadBalancer`

Como podemos observar en el diagrama, la clase `PenaltyBasedLoadBalancer` hereda de la clase abstracta `LoadBalancer`, que define la interfaz común a todos los algoritmos de balanceado de carga.

Asimismo, los objetos `PenaltyBasedLoadBalancer` utilizan un objeto `ClusterServerDBConnector` para obtener los servidores de máquinas virtuales candidatos y el estado de los mismos. Los objetos `ClusterServerDBConnector` permiten manipular la base de datos del servidor de *cluster*.

Finalmente, los parámetros de configuración del algoritmo de balanceado de carga se fijarán en el fichero de configuración del demonio del servidor de *cluster*.

4.5.7.2. Reactores del servidor de *cluster*

El demonio del servidor de *cluster* deberá procesar los paquetes enviados por los servidores de máquinas virtuales activos y los paquetes enviados por el repositorio de imágenes.

Asimismo, también deberá procesar los eventos de conexión, desconexión y reconexión generados por la red junto con los paquetes enviados por el servidor web. A través de estos últimos, el servidor de *cluster* recibirá las peticiones de los usuarios.

Para aumentar la claridad del código fuente y simplificar la corrección de errores, hemos definido cuatro clases:

- `EndpointPacketReactor`, cuyos métodos procesan los paquetes enviados por el servidor web.

- `ImageRepositoryPacketReactor`, cuyos métodos procesan los paquetes enviados por el repositorio de imágenes.
- `NetworkEventsReactor`, cuyos métodos procesan los eventos de conexión, desconexión y reconexión generados por la red.
- `VMServerPacketReactor`, cuyos métodos procesan los paquetes enviados por los servidores de máquinas virtuales.

En esta sección, mostraremos las principales relaciones de todas ellas.

4.5.7.2.1. La clase `EndpointPacketReactor`

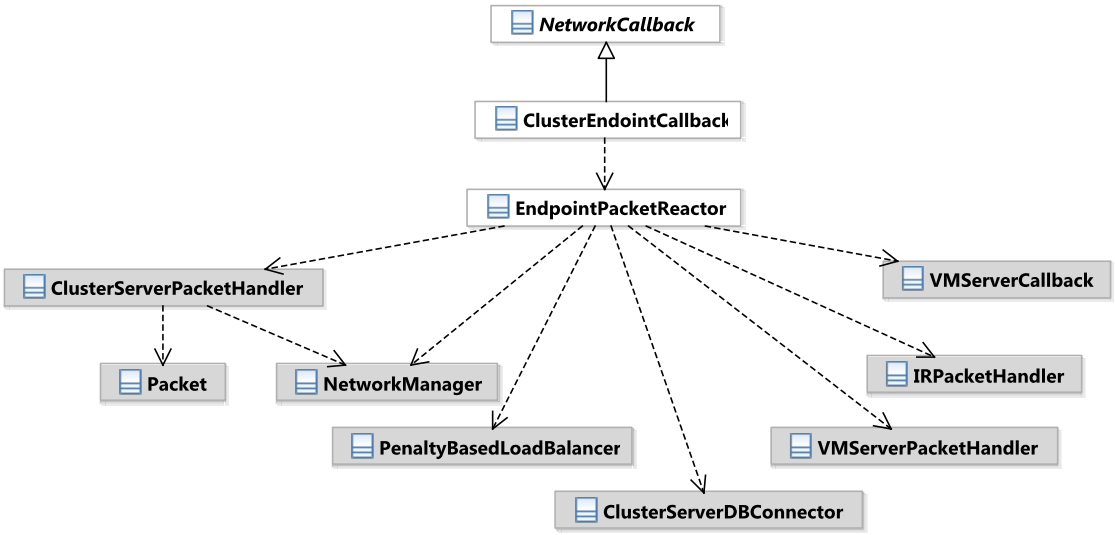

FIGURA 4.67: Principales relaciones de la clase `EndpointPacketReactor`

Las instancias de la clase `EndpointPacketReactor` se utilizan para procesar los paquetes enviados por el servidor web. El diagrama de clases de la figura 4.67 muestra las principales relaciones de esta clase.

Como podemos observar en el diagrama, los objetos `EndpointPacketReactor` procesan los paquetes enviados por el servidor web a través de una instancia de la clase `ClusterEndpointCallback`, que hereda de la clase `NetworkCallback`.

Además, los objetos `EndpointPacketReactor` utilizan

- un objeto `ClusterServerPacketHandler` para leer y generar los paquetes que intercambian con el servidor de *cluster*.
- un objeto `VMServerPacketHandler` para generar los paquetes que envían a los servidores de máquinas virtuales.
- un objeto `VMServerCallback`, que se utilizará durante el establecimiento de la conexión con un servidor de máquinas virtuales.
- un objeto `ImageRepositoryPacketHandler` para generar los paquetes que envían a los servidores de máquinas virtuales.

Para enviar esos paquetes, los objetos `EndpointPacketReactor` utilizan un objeto `NetworkManager`.

Por otra parte, los objetos `EndpointPacketReactor` ejecutarán el algoritmo de balanceado de carga cuando sea preciso. Por ello, utilizarán un objeto `PenaltyBasedLoadBalancer`. Es importante notar que, para facilitar el cambio del algoritmo de balanceado de carga, este objeto sólo se manipulará a través de la interfaz definida por la clase `LoadBalancer`.

Finalmente, los objetos `EndpointPacketReactor` utilizan un objeto `ClusterServerDBConnector` para manipular la base de datos del servidor de *cluster*.

4.5.7.2.2. La clase `ImageRepositoryPacketReactor`

Las instancias de la clase `ImageRepositoryPacketReactor` procesan los paquetes enviados por el repositorio de imágenes. El diagrama de clases de la figura 4.68 muestra las principales relaciones de esta clase.

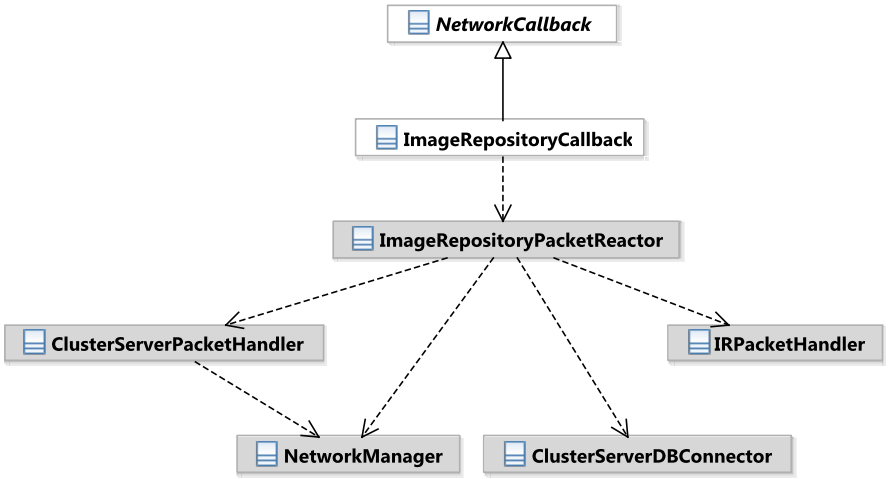

FIGURA 4.68: Principales relaciones de la clase `ImageRepositoryPacketReactor`

En primer lugar, para procesar los paquetes procedentes del repositorio de imágenes, se utiliza una instancia de la clase `ImageRepositoryCallback`, que hereda de la clase `NetworkCallback`. Esos objetos utilizan una instancia de la clase `ImageRepositoryPacketReactor` para procesar los paquetes recibidos.

Al igual que sucede con los objetos `EndpointPacketReactor`, los objetos `ImageRepositoryPacketHandler` utilizan

- un objeto `ClusterServerPacketHandler` para construir los paquetes que se enviarán al servidor web,
- un objeto `ImageRepositoryPacketHandler` para leer los paquetes que se intercambian con el repositorio de imágenes,
- un objeto `NetworkManager` para enviar paquetes, y
- un objeto `ClusterServerDBConnector` para manipular la base de datos del servidor de *cluster*.

A diferencia de lo que ocurre en el caso de la clase `EndpointPacketReactor`, los objetos `ImageRepositoryPacketReactor` no necesitan comunicarse con un servidor de máquinas virtuales. Por ello, no utilizan un objeto `VMServerPacketHandler`.

Asimismo, estos objetos no establecen conexiones con los servidores de máquinas virtuales, por lo que no utilizan ningún objeto `VMServerCallback`.

4.5.7.2.3. La clase `VMServerPacketReactor`

Los objetos `VMServerPacketReactor` procesan los paquetes enviados por los servidores de máquinas virtuales. El diagrama de clases de la figura muestra las principales relaciones de la clase `VMServerPacketReactor`.

Como podemos observar en el diagrama de clases, al igual que sucede con los objetos `EndpointPacketReactor`, los objetos `VMServerPacketHandler` utilizan

- un objeto `ClusterServerPacketHandler` para generar los paquetes que enviarán al servidor web.
- un objeto `VMServerPacketHandler` para leer los paquetes que se intercambian con los servidores de máquinas virtuales.
- un objeto `NetworkManager` para enviar paquetes.
- un objeto `ClusterServerDBConnector` para manipular la base de datos del servidor de *cluster*.

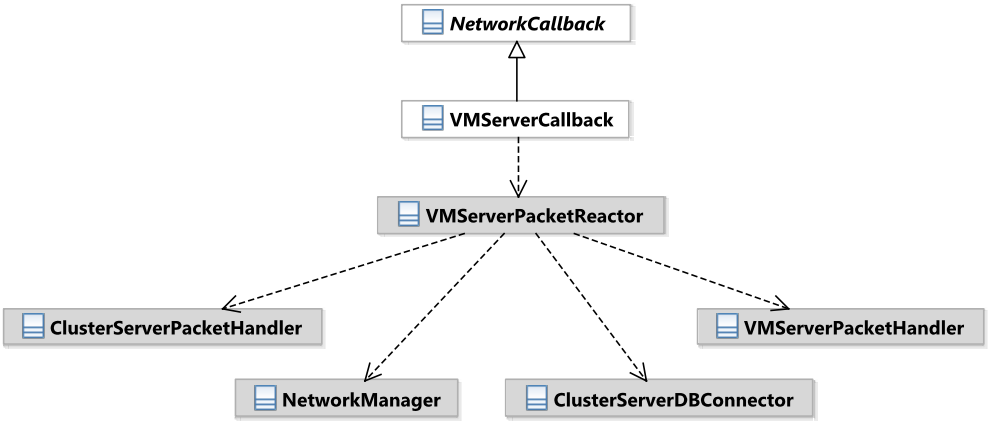

FIGURA 4.69: Principales relaciones de la clase `VMServerPacketReactor`

Finalmente, como estos objetos nunca envían paquetes al repositorio de imágenes, no necesitan utilizar un objeto `ImageRepositoryPacketHandler`. Asimismo, tampoco utilizan un objeto `VMServerCallback`, ya que nunca crean conexiones con los servidores de máquinas virtuales.

4.5.7.2.4. La clase `NetworkEventsReactor`

Los objetos `NetworkEventsReactor` tratan los eventos de conexión, reconexión y desconexión de máquinas generados por la red. El diagrama de clases de la figura 4.70 muestra las principales relaciones de la clase `NetworkEventsReactor`.

Como podemos observar en la figura 4.70, los objetos `VMServerCallback` e `ImageRepositoryCallback` utilizan un objeto `NetworkEventsReactor` para procesar los eventos asociados a las conexiones de red.

Para procesar esos eventos, los objetos `NetworkEventsReactor` se limitan a actualizar el estado de la máquinas correspondiente en la base de datos del servidor de *cluster*. Para ello, utilizan un objeto `ClusterServerDBConnector`.

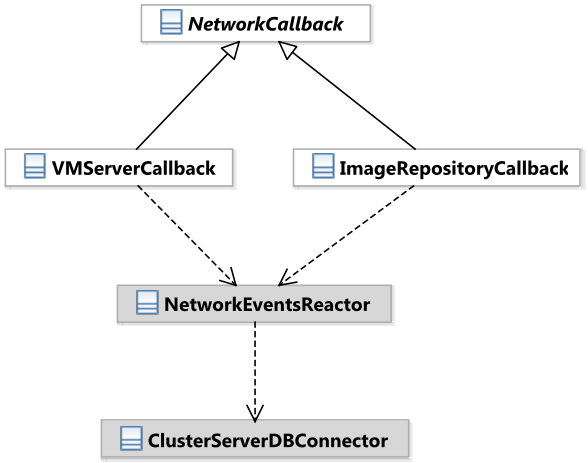

FIGURA 4.70: Principales relaciones de la clase `NetworkEventsReactor`

4.5.7.3. La clase `ClusterServerMainReactor`

La clase `ClusterServerMainReactor` es la clase principal del demonio del servidor de *cluster*. Sus principales responsabilidades son las siguientes:

- realizar los procesos de arranque y apagado del servidor de *cluster*, incluyendo la configuración de los reactores de paquetes y el establecimiento de la conexión con el repositorio de imágenes.
- monitorizar los comandos de arranque de máquinas virtuales, generando errores por *timeout* cuando sea necesario.

El diagrama de clases de la figura 4.71 muestra las principales relaciones de la clase `ClusterServerMainReactor`.

Durante el proceso de inicialización, el objeto `ClusterServerMainReactor` crea y configura la base de datos. Para ello, utiliza un objeto `DBConfigurator`. La clase `DBConfigurator` se encuentra en el paquete `ccutils`.

Por otra parte, antes de crear cualquier conexión de red es necesario

- instanciar un objeto `NetworkManager`.
- instanciar los objetos que manipularán los paquetes recibidos a través de las distintas conexiones de red. Por ello, el objeto `ClusterServerMainReactor` instancia las clases `ImageRepositoryPacketHandler`, `ClusterServerPacketHandler` y `VMServerPacketHandler`.
- crear los objetos que procesarán los paquetes recibidos por la misma. Por ello, el objeto `ClusterServerMainReactor` instancia las clases `VMServerPacketReactor`, `EndpointPacketReactor`, `ImageRepositoryPacketReactor`, `ClusterEndpointCallback`, `VMServerCallback` e `ImageRepositoryCallback`.
- crear los objetos que procesarán los eventos de conexión, reconexión y desconexión generados por la red. Por ello, el objeto `ClusterServerMainReactor` instancia un objeto `NetworkEventsReactor`.

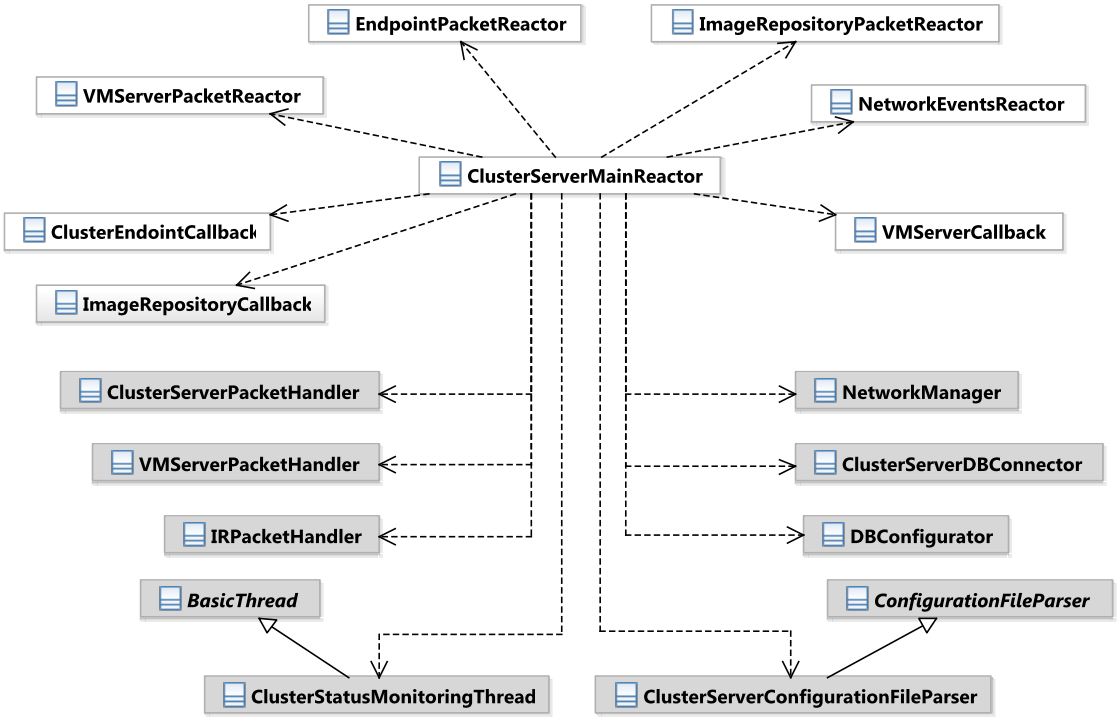

FIGURA 4.71: Principales relaciones de la clase `ClusterServerMainReactor`

Además, también hemos visto que, para procesar los paquetes recibidos y los eventos generados por la red, es necesario manipular la base de datos del servidor de *cluster*. Por ello, los objetos `ClusterServerMainReactor` también instancian un objeto `ClusterServerDBConnector` durante el proceso de inicialización.

Y al igual que ocurre en el servidor de máquinas virtuales y en el repositorio de imágenes, durante el proceso de inicialización del demonio del servidor de *cluster* también se procesa el contenido de un fichero de configuración. Para ello, se utiliza una instancia de la clase `ClusterServerConfigurationFileParser`, que hereda de la clase `ConfigurationFileParser` del paquete `ccutils`.

Finalmente, la clase `ClusterStatusMonitoringThread` se corresponde con el hilo que envía periódicamente los paquetes de solicitud de estado a todas las máquinas del *cluster*. Como la gran mayoría de las clases de hilo que utilizamos, la clase `ClusterStatusMonitoringThread` hereda de la clase `BasicThread`, definida también en el paquete `ccutils`.

4.5.7.4. Arranque del servidor de *cluster*: secuencia básica

En esta sección, mostraremos el proceso de arranque del demonio del servidor de *cluster*. Para ello, haremos uso del diagrama de secuencia de la figura 4.72. Por claridad, en dicho diagrama

- estamos asumiendo que no se produce ningún error,
- hemos omitido la creación de los objetos `VMServerPacketHandler`, `ImageRepositoryPacketHandler` y `ClusterServerPacketHandler`, y
- hemos omitido la inicialización de los objetos `VMServerCallback`, `ImageRepositoryCallback` y `ClusterEndpointCallback`.

Como podemos observar en el diagrama de la figura 4.72, en el arranque del demonio del servidor de *cluster* se distinguen las siguientes fases:

1. se procesa el fichero de configuración del servidor de *cluster*.

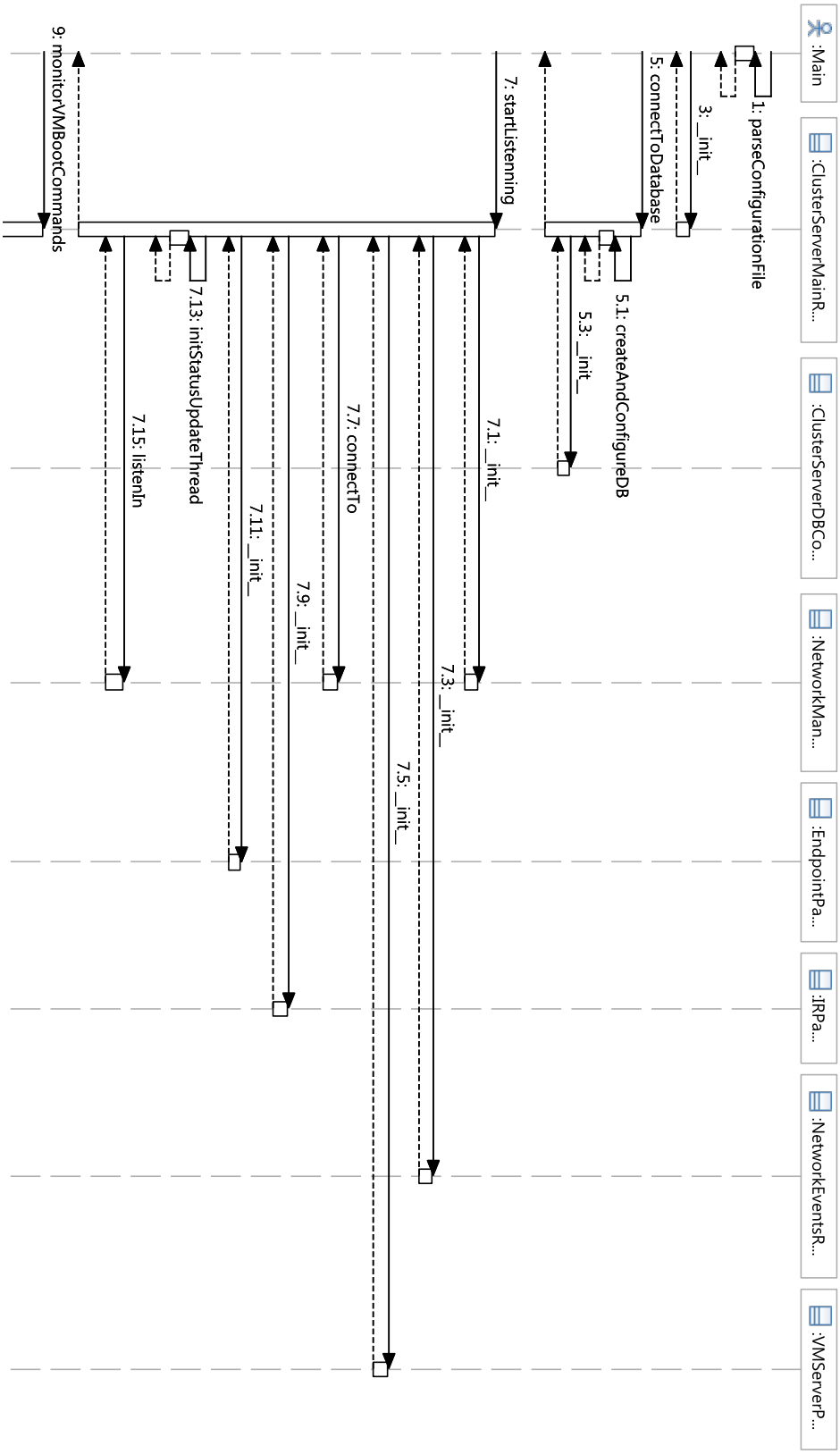

FIGURA 4.72: Arranque del servidor de *cluster*: secuencia básica

2. se configura la base de datos del servidor de *cluster*.
3. se prepara todo lo necesario para establecer la conexión con el repositorio de imágenes. Acto seguido, se intenta crear dicha conexión.
4. se prepara todo lo necesario para poder establecer conexiones con los servidores de máquinas virtuales.
5. se prepara todo lo necesario para poder procesar los paquetes enviados desde el servidor web.
6. se crea e inicia el hilo que envía periódicamente los paquetes de actualización de estado a todas las máquinas del *cluster*.
7. se crea la conexión por la que el servidor de *cluster* atenderá las peticiones del servidor web.
8. el hilo principal inicia la monitorización de los comandos de arranque de máquinas virtuales.

4.5.7.5. Arranque del servidor de *cluster*: tratamiento de errores

Durante el proceso de arranque del servidor de *cluster*, se producirán errores si

- el contenido del fichero de configuración del servidor de *cluster* es incorrecto.
- no se puede establecer la conexión de red con el repositorio de imágenes.
- la configuración de la base de datos falla.
- el puerto en el que escucha el demonio del servidor de *cluster* ya está en uso.

Para tratar todos estos errores, se procede de forma similar:

1. el hilo principal termina inmediatamente de monitorizar el estado de los comandos de arranque.
2. acto seguido, desde el hilo principal
   a) se cierran ordenadamente las conexiones de red que hayan sido establecidas. A partir de este momento, el servidor de *cluster* dejará de atender peticiones.
   b) se detiene el servicio de red.
   c) se detiene el hilo que envía periódicamente los paquetes de solicitud de estado al resto de máquinas del *cluster*.

Finalmente, debemos recordar que, para evitar cuelgues, sólo es posible detener el servicio de red desde el hilo principal. Por ello, las tareas de apagado se realizan en dicho hilo.

4.5.7.6. Gestión de los servidores de máquinas virtuales

En esta sección, estudiaremos todas las interacciones que tienen lugar entre el servidor web y el servidor de *cluster* para

- realizar las altas y bajas de los servidores de máquinas virtuales del *cluster*,
- arrancar y apagar los servidores de máquinas virtuales del *cluster*, y
- modificar la configuración básica de los servidores de máquinas virtuales del *cluster*.

Como viene siendo habitual, todas las interacciones entre el servidor web y el servidor de *cluster* se basan en el intercambio de diversos tipos de paquete entre ambas máquinas.

Por otra parte, para facilitar la comprensión de todas las secuencias, de ahora en adelante siempre empezaremos explicando la secuencia básica. Posteriormente, mostraremos las secuencias alternativas, que se corresponden con el tratamiento de errores.

4.5.7.6.1. Alta de un servidor de máquinas virtuales: secuencia básica

Para dar de alta un servidor de máquinas virtuales, el servidor web envía un paquete del tipo Registrar servidor de máquinas virtuales al servidor de *cluster*. Estos paquetes contienen

- la dirección IP y el puerto de escucha del demonio residente en el servidor de máquinas virtuales que se desea registrar,
- el nombre que se desea dar al nuevo servidor de máquinas virtuales,
- un *flag* que indica si el servidor de máquinas virtuales se usará preferentemente para crear o editar imágenes de disco, y
- el identificador único de la petición.

En el servidor de *cluster*, este paquete se procesará de acuerdo al diagrama de secuencia de la figura 4.73. Como podemos observar en dicho diagrama, en el proceso de registro del servidor de máquinas virtuales se siguen los siguientes pasos:

1. se comprueba si el nombre y la dirección IP del servidor de máquinas virtuales a registrar ya están en uso. Este paso es necesario porque todo servidor de máquinas virtuales registrado en un mismo servidor de *cluster* debe tener un nombre y una dirección IP únicos.
2. se establece la conexión con el servidor de máquinas virtuales.

   Aunque no se aprecia en el diagrama de la figura 4.73, al crear las conexiones con los servidores de máquinas virtuales aprovechamos una de las optimizaciones de la red, el control de la concurrencia, de la que hablamos en la sección 4.5.3.6.1.

   Así, el mismo objeto procesará los paquetes enviados por todos los servidores de máquinas virtuales y, con independencia del número de conexiones que el servidor de *cluster* mantenga con los servidores de máquinas virtuales, el número de hilos de red permanecerá constante.

   Esto resulta imprescindible para que la escalabilidad del sistema no se vea comprometida, y también explica por qué

   - todos los servidores de máquinas virtuales dados de alta en el servidor de *cluster* tengan una dirección IP única, y por qué
   - algunos paquetes enviados por los servidores de máquinas virtuales contengan la IP del remitente, tal y como vimos en la sección 4.5.6.14.

     Tras la incorporación del modo *unicast* de a las clases del paquete `network`, ya no es necesario que estos paquetes contengan la dirección IP del remitente. No obstante, para garantizar la estabilidad de la versión final, hemos decidido seguir utilizándolo.
3. cuando se establece la conexión, se registra el servidor de máquinas virtuales en la base de datos del servidor de *cluster*.
4. finalmente, se envía un paquete del tipo Comando ejecutado al servidor de *cluster*. Este contiene el identificador único de la petición.

Tras finalizar este proceso, puede parecer que ya es posible utilizar el servidor de máquinas virtuales. Pero, como hemos visto, el algoritmo de balanceado de carga necesita conocer el estado de los servidores de máquinas virtuales para funcionar. Por ello, el servidor de máquinas virtuales sólo podrá utilizarse después de que el servidor de *cluster* haya averiguado su estado. Eso ocurrirá durante la siguiente actualización del estado del *cluster*.

Así, como puede observarse en la parte inferior del diagrama de secuencia de la figura 4.73, el proceso de registro del servidor de máquinas virtuales finaliza tras la recepción del paquete con el estado de dicha máquina.

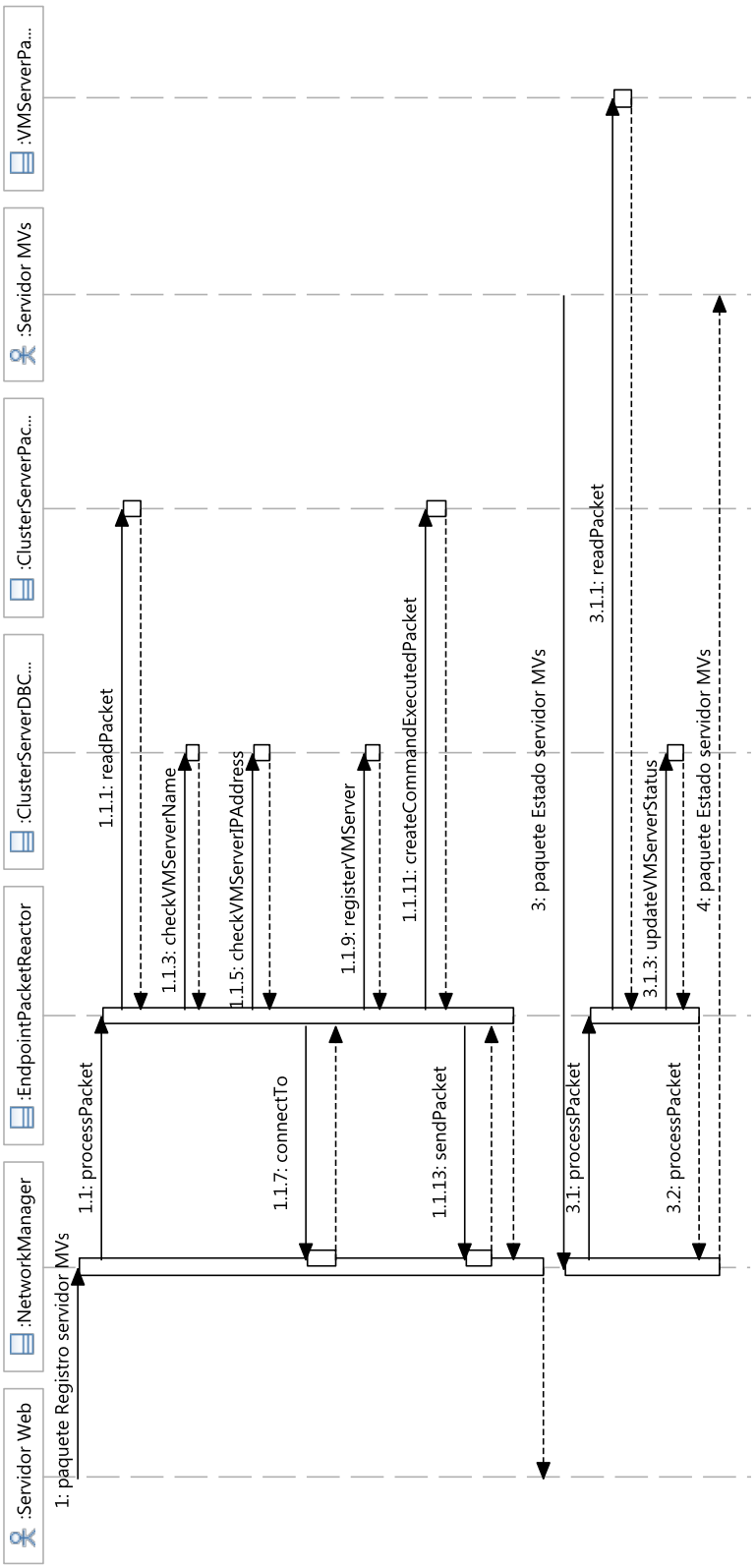

FIGURA 4.73: Alta de un servidor de máquinas virtuales: secuencia básica

4.5.7.6.2. Alta de un servidor de máquinas virtuales: tratamiento de errores

Al dar de alta un servidor de máquinas virtuales, se producirán errores cuando

- el nombre o la IP del servidor de máquinas virtuales ya están en uso, o cuando
- no es posible establecer la conexión con el servidor de máquinas virtuales.

Ambos errores se procesan de forma idéntica:

1. se construye un paquete del tipo Error de registro de servidor de máquinas virtuales. Estos paquetes contienen el identificador único de la petición y un código de descripción del error.
2. se envía este paquete al servidor *web*.

Es importante notar que,

- debido a que la actualización de la base de datos se hace en último lugar, siempre que se produzcan errores no es necesario borrar nada de la base de datos del servidor de *cluster*.
- para evitar la aparición de problemas de coherencia, al dar de alta un servidor de máquinas virtuales se asume que este no contiene ninguna imagen de disco.

4.5.7.6.3. Arranque de un servidor de máquinas virtuales: secuencia básica

Para arrancar un servidor de máquinas virtuales ya registrado, el servidor web enviará al servidor de *cluster* un paquete del tipo Arrancar servidor de máquinas virtuales. Este paquete contiene

- el nombre o la dirección IP del servidor de máquinas virtuales a arrancar, y
- el identificador único de la petición.

Los paquetes del tipo Arrancar servidor de máquinas virtuales se procesan prácticamente igual que los paquetes del tipo Registrar servidor de máquinas virtuales, y sólo existen dos diferencias:

- antes de establecer la conexión con el servidor de máquinas virtuales, los datos de conexión se extraen de la base de datos. Además, puesto que ya están almacenados en ella, los datos de conexión no se insertan en la base de datos del servidor de *cluster*.
- tras establecer la conexión con el servidor de máquinas virtuales,
  - se le envía un paquete del tipo Solicitud de identificadores de máquinas virtuales activas. Esto hace posible que, tras una caída de un servidor de máquinas virtuales, las máquinas virtuales activas que este alberga vuelvan a registrarse en la infraestructura.
  - se actualizan las imágenes de disco almacenadas en el servidor de máquinas virtuales. Para ello, se le envían los paquetes de los tipos Borrar imagen y Desplegar imagen que sean necesarios. Justificaremos la necesidad de este paso en las secciones y .

4.5.7.6.4. Arranque de un servidor de máquinas virtuales: tratamiento de errores

Durante el arranque de un servidor de máquinas virtuales pueden producirse errores cuando

- el servidor de máquinas virtuales que se pretende arrancar no está registrado, o cuando
- no es posible establecer la conexión con dicho servidor de máquinas virtuales.

Nuevamente, estos errores se procesan casi igual que los errores de registro de un servidor de máquinas virtuales.
Lo único que cambia en este caso es el tipo del paquete de error: tras detectar un error en el proceso de arranque de un servidor de máquinas virtuales, el servidor de *cluster* envía al servidor web un paquete del tipo Error en el arranque de servidor de máquinas virtuales, que contiene, nuevamente, el identificador único de la petición y un código de descripción del error.

4.5.7.6.5. Apagado de un servidor de máquinas virtuales: secuencia básica

Para apagar un servidor de máquinas virtuales, el servidor web enviará al servidor de *cluster* un paquete del tipo Apagar o dar de baja servidor de máquinas virtuales. Estos paquetes contienen

- el nombre o la dirección IP del servidor de máquinas virtuales a apagar,
- un *flag*, que toma el valor False, y
- el identificador único de la petición.

Cuando se reciben, el servidor de *cluster* los procesa de acuerdo al diagrama de secuencia de la figura 4.74. Como podemos observar en ese diagrama, se siguen los siguientes pasos:

1. se extraen los datos del servidor de máquinas virtuales de la base de datos del servidor de *cluster*.
2. se actualiza el estado del servidor de máquinas virtuales en la base de datos del servidor de *cluster*. A partir de este momento, el algoritmo de balanceado de carga dejará de contar con él.
3. se construye y envía un paquete del tipo Apagado con espera al servidor de máquinas virtuales.
4. se cierra la conexión con el servidor de máquinas virtuales.
5. se construye un paquete del tipo Comando ejecutado y se envía al servidor web.

4.5.7.6.6. Apagado de un servidor de máquinas virtuales: tratamiento de errores

Cuando se apaga un servidor de máquinas virtuales, se producirán errores si

- el servidor de máquinas virtuales no está registrado.
- se ha perdido la conexión con el servidor de máquinas virtuales.

Estos errores se procesarán casi igual que los que estudiamos en la sección 4.5.7.6.4. Naturalmente, en este caso no se enviarán al servidor de *cluster* paquetes del tipo Error en el arranque de servidor de máquinas virtuales, sino paquetes del tipo Error en el apagado o baja de servidor de máquinas virtuales.

Finalmente, es importante notar que, si se intenta apagar un servidor de máquinas virtuales que ya está apagado, la petición no tendrá ningún efecto. Para ahorrar ancho de banda, hemos decidido no generar errores en estas situaciones.

4.5.7.6.7. Baja de un servidor de máquinas virtuales: secuencia básica

El proceso de baja de un servidor de máquinas virtuales es muy similar al proceso de apagado, del que hablamos en la sección 4.5.7.6.5. Las modificaciones que hay que realizar en dicho proceso son las siguientes:

- sólo se enviará el paquete de apagado al servidor de máquinas virtuales si este no está apagado.
- en lugar de actualizar el estado del servidor de máquinas virtuales en la base de datos del servidor de *cluster*, se borra toda la información asociada al servidor de máquinas virtuales de dicha base de datos.

Por otra parte, en la sección 4.5.7.6.5 dijimos que los paquetes del tipo Apagar o dar de baja servidor de máquinas virtuales contienen un *flag*. En las operaciones de apagado, este *flag* toma el valor `False`. En las de baja, toma el valor `True`. Así, aunque estas dos operaciones se solicitan mediante paquetes del mismo tipo, el *flag* nos permite distinguir entre ambas.

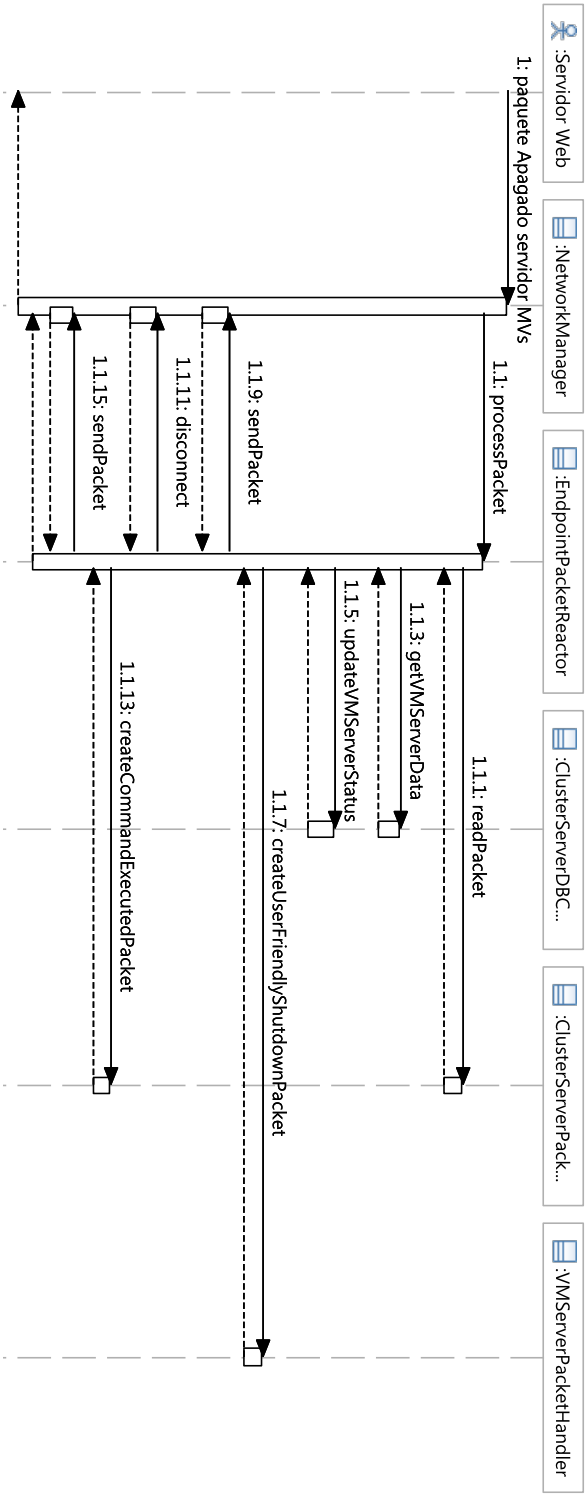

FIGURA 4.74: Apagado de un servidor de máquinas virtuales: secuencia básica

4.5.7.6.8. Baja de un servidor de máquinas virtuales: tratamiento de errores

Cuando se da de baja un servidor de máquinas virtuales, sólo se producirán errores cuando el servidor de máquinas virtuales que se especifica en la petición no está registrado.

Para tratar estos errores, se procede exactamente igual que en el apagado de un servidor de máquinas virtuales. Puesto que en la sección 4.5.7.6.6 vimos este proceso en detalle, no volveremos a reproducirlo.

4.5.7.6.9. Modificación de la configuración de un servidor de máquinas virtuales: secuencia básica

Para facilitar la administración del sistema, es posible modificar la configuración básica de un servidor de máquinas virtuales (su nombre, su dirección IP, su puerto y su uso para crear o editar imágenes) sin necesidad de darlo de baja y volverlo a registrar.

Para ello, el servidor web enviará un paquete del tipo Cambiar configuración de servidor al servidor de máquinas virtuales. Estos paquetes contienen

- el nuevo nombre del servidor de máquinas virtuales,
- la nueva dirección IP y el nuevo puerto del servidor de máquinas virtuales,
- el nuevo valor del *flag* que indica si el servidor de máquinas virtuales se podrá utilizar para crear o editar imágenes, y
- el identificador único de la petición.

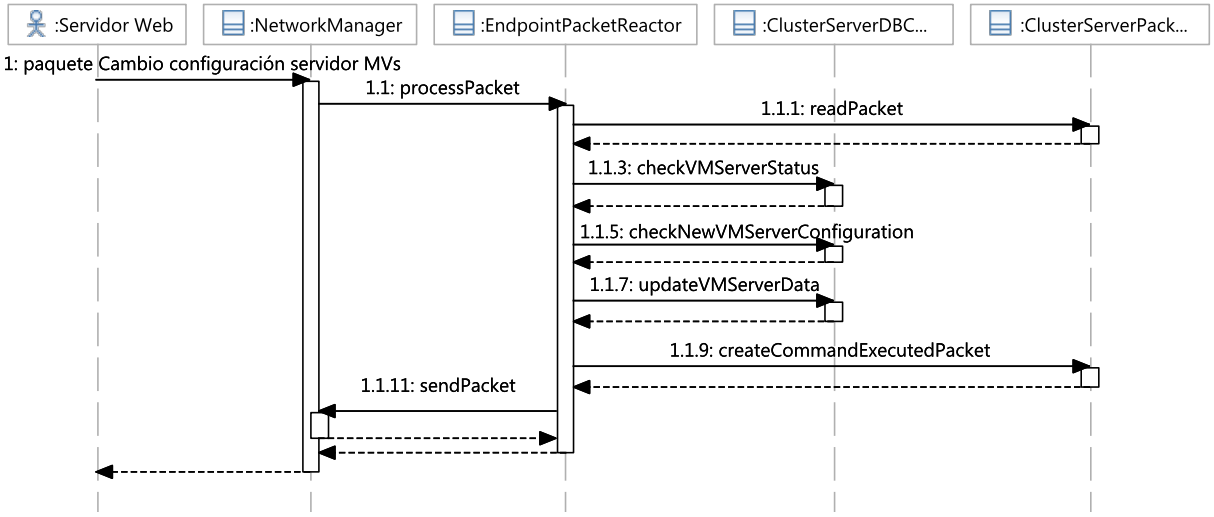

FIGURA 4.75: Modificación de la configuración de un servidor de máquinas virtuales: secuencia básica

Cuando se recibe uno de estos paquetes, el servidor de *cluster* lo procesa de acuerdo al diagrama de secuencia de la figura 4.75. Como podemos observar en dicho diagrama, tras recibir y leer el contenido del paquete, se siguen los siguientes pasos:

1. se comprueba que el servidor de máquinas virtuales está apagado. Sólo es posible modificar la configuración de los servidores de máquinas virtuales apagados.
2. se comprueba que la nueva configuración del servidor de máquinas virtuales es correcta. Para ello, el nuevo nombre y la nueva dirección IP del servidor de máquinas virtuales no deben estar en uso.

3. se modifica la configuración del servidor de máquinas virtuales en la base de datos del servidor de *cluster*.

4. se envía al servidor web un paquete del tipo Comando ejecutado. Con él, se indica el fin del procesamiento de la petición.

4.5.7.6.10. Modificación de la configuración de un servidor de máquinas virtuales: tratamiento de errores

Al modificar la configuración de un servidor de máquinas virtuales, se producirán errores cuando

- el nuevo nombre o la nueva dirección IP del servidor de máquinas virtuales ya están en uso, o cuando
- el servidor de máquinas virtuales está arrancado, o cuando
- el servidor de máquinas virtuales no está registrado.

Todos estos errores se procesan igual: tan pronto como se detectan,

1. se aborta la modificación de la configuración del servidor de máquinas virtuales.

2. se genera un paquete del tipo, que contiene el identificador único de la petición y un código de descripción del error.

3. se envía este paquete al servidor web.

4.5.7.7. Gestión básica de máquinas virtuales

En esta sección, mostraremos cómo interactúan el servidor web y el servidor de *cluster* en los procesos de arranque, apagado forzoso y reinicio forzoso de una máquina virtual. Estas interacciones son las más frecuentes.

Por claridad, en todos los casos empezaremos mostrando la secuencia básica. Posteriormente, mostraremos las secuencias alternativas, que se corresponden con el tratamiento de errores.

4.5.7.7.1. Arranque de una máquina virtual: secuencia básica

Cuando un usuario solicita la creación de una máquina virtual, el servidor web envía al servidor de *cluster* un paquete del tipo Arrancar máquina virtual. Este paquete contiene

- el identificador del usuario que ha enviado la petición,
- el identificador de la imagen que utilizará la máquina virtual para arrancar, y
- el identificador único de la petición.

El diagrama de secuencia de la figura 4.76 muestra cómo se procesan estos paquetes en el servidor de *cluster*. En él, podemos distinguir estos pasos:

1. se lee el contenido del paquete de arranque.

2. se ejecuta el algoritmo de balanceado de carga para averiguar cuál es el servidor de máquinas virtuales que alojará la nueva máquina virtual.

3. se reservan los recursos que consumirá la máquina virtual. En la próxima sección, justificaremos detalladamente la necesidad de este paso.

4. se construye un paquete del tipo Crear máquina virtual, y acto seguido se enviará al servidor de máquinas virtuales que ha escogido el algoritmo de balanceado de carga.

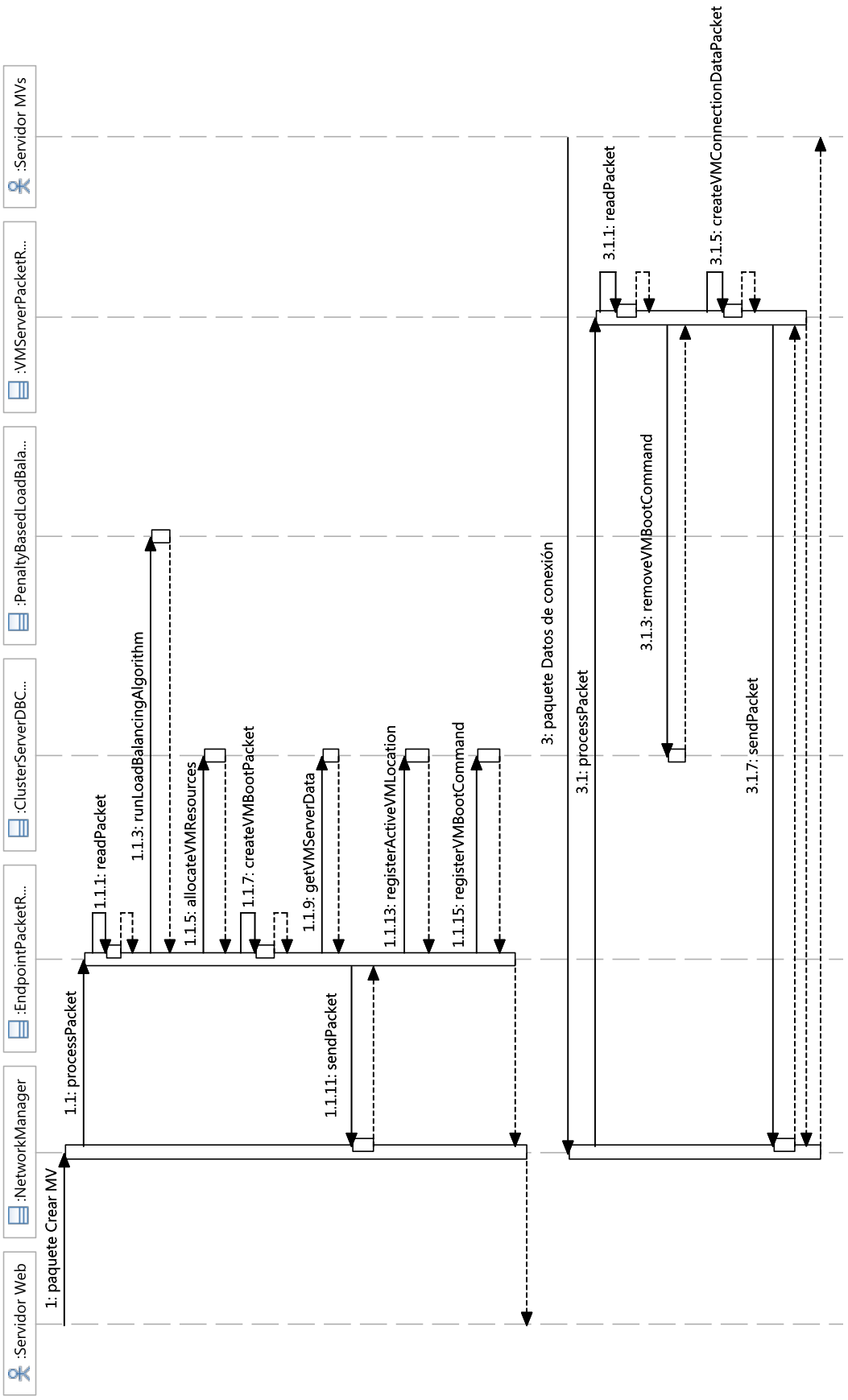

FIGURA 4.76: Arranque de una máquina virtual: secuencia básica

5. se registra la petición de arranque en la base de datos del servidor de *cluster*. En la sección 4.5.7.7.3 justificaremos este paso.

6. se registra la ubicación de la nueva máquina virtual. Esta información permitirá ahorrar mucho ancho de banda en los procesos de arranque y apagado forzoso de máquinas virtuales.

7. cuando ese servidor de máquinas virtuales acaba de crear la máquina virtual, envía un paquete del tipo Datos de conexión de MV.

8. durante el procesamiento de ese paquete,

   a) se borrará la petición de arranque de la base de datos del servidor de *cluster*.

   b) se construirá un paquete del tipo Datos de conexión de MV y se enviará al servidor web. Estos paquetes contienen

   - la dirección IP, el puerto y la contraseña del servidor VNC asociado a la nueva máquina virtual, y
   - el identificador único de la petición.

4.5.7.7.2. Tratamiento de las ráfagas de peticiones

Para conocer el estado de los servidores de máquinas virtuales y del repositorio de imágenes, el servidor de *cluster* debe enviar paquetes del tipo Solicitud de estado a los servidores de máquinas virtuales.

Ahora bien, para no desperdiciar el ancho de banda de la red, no es posible enviar continuamente estos paquetes. Y aunque lo hiciésemos, las respuestas de los servidores de máquinas virtuales tardan cierto tiempo en recibirse y en procesarse. Ese tiempo no es despreciable.

Por tanto, si no introducimos cambios, no será posible conocer en tiempo real el estado de los servidores de máquinas virtuales y el estado del repositorio de imágenes. Por ejemplo, supongamos que

- el *cluster* cuenta con n servidores de máquinas virtuales activos, que
- cada servidor de máquinas virtuales puede albergar cuatro máquinas virtuales, que
- el estado de las máquinas virtuales se actualiza una vez cada segundo, y que
- en cierto momento del día m alumnos, con $4 < m \leq 4 \cdot n$, solicitan a la vez el arranque de una máquina virtual.

Entre dos actualizaciones consecutivas de la base de datos de estado, el algoritmo de balanceado de carga puede ejecutarse varias veces, ya que esas actualizaciones no son inmediatas. Por ello puede ocurrir que, al atender las peticiones de la ráfaga, el algoritmo de balanceado de carga coloque más de cuatro máquinas virtuales en uno o más servidores de máquinas virtuales.

Cuando estos servidores procesan las peticiones que les ha enviado el servidor de *cluster*, detectarán un error. Por tanto, a pesar de que hay suficientes recursos, un número arbitrario de peticiones de arranque fallarán.

Si bien es cierto que los alumnos pueden volver a emitir las peticiones,

- se está desperdiciando ancho de banda emitiendo peticiones que, sin duda, fallarán, y
- en los servidores de máquinas virtuales se están desperdiciando recursos, penalizándose el rendimiento de las máquinas virtuales activas que albergan.

Este problema se agrava cuando se crean o editan imágenes, ya que las transferencias FTP entre los servidores de máquinas virtuales y el repositorio de imágenes consumen mucho más ancho de banda.

Para resolver este problema, tal y como aparece en el diagrama de secuencia de la figura 4.76, antes de enviar una petición se reservan los recursos que esta consumirá en el servidor de máquinas virtuales que tiene asignado y en el repositorio de imágenes.

Como veremos en la sección 4.5.7.8, cuando se reciba el paquete con el estado de ese servidor, los preasignados se liberarán.

Es importante notar que, tal y como dijimos en la la sección 4.5.3.5.2, los paquetes recibidos por una misma conexión de red se procesan de uno en uno. Esto, junto con el mecanismo que acabamos de explicar, nos permite garantizar que no se producirán problemas al recibir ráfagas de peticiones en el servidor de *cluster*.

4.5.7.7.3. Arranque de una máquina virtual: tratamiento de errores

Durante el arranque de una máquina virtual, se producirán errores si

- las imágenes de disco correspondientes no existen.
- no existe un servidor de máquinas virtuales que la pueda albergar.
- no es posible establecer la conexión con el servidor de máquinas virtuales.
- el proceso de arranque de la máquina virtual falla en el servidor de máquinas virtuales.

Los dos primeros errores se detectan examinando la salida del algoritmo de balanceado de carga. Para procesarlos, se procede de la misma manera:

1. se construye un paquete del tipo Error de arranque de máquina virtual. Estos contienen el identificador único de la petición y un código de descripción del error.
2. se envía dicho paquete al servidor web.

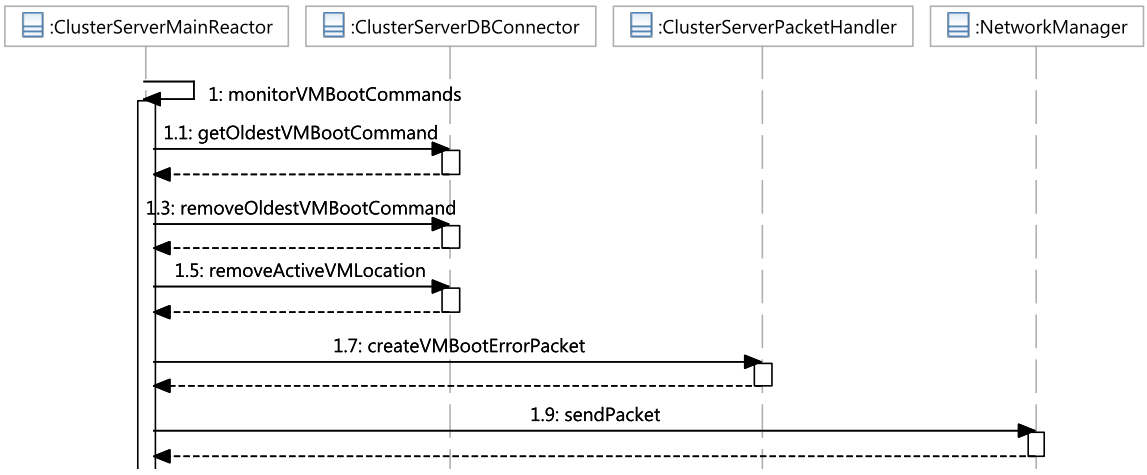

FIGURA 4.77: Monitorización de los comandos de arranque de máquinas virtuales

Por otra parte, el servidor de *cluster* detectará los dos últimos errores utilizando *timeouts*. Para ello, el hilo principal monitorizará todos los comandos de arranque. Dicha monitorización se realiza de acuerdo al diagrama de secuencia de la figura 4.77, en el que se distinguen los siguientes pasos:

1. el hilo principal extrae de la base de datos el identificador de la petición de arranque de máquina virtual más antigua que cumple

$$\text{tiempo espera} \geq \text{tiempo máximo}$$

   El tiempo máximo se lee del fichero de configuración del demonio del servidor de *cluster*.

2. se elimina la ubicación de la máquina virtual correspondiente.

3. acto seguido, se construye un paquete del tipo Error de arranque de máquina virtual y se envía al servidor de *cluster*.

Es importante notar que

- cuando se detecta un error al examinar la salida del algoritmo de balanceado de carga, no se añade el identificador de la petición a la base de datos.

- cuando se reciben los datos de conexión al servidor VNC de la máquina virtual, se borrará el identificador de la petición de arranque de la base de datos.

Así, no se generarán errores por *timeout* cuando se detectan los dos primeros tipos de error o cuando el arranque de la máquina virtual se realiza con normalidad.

4.5.7.7.4. Apagado forzoso de una máquina virtual: secuencia básica

Los usuarios pueden forzar el apagado de una máquina virtual cuando esta se cuelga. En estos casos, el servidor web envía un paquete del tipo Destrucción de máquina virtual al servidor de *cluster*. Estos paquetes contienen

- el identificador único de la petición, y

- el identificador único de la máquina virtual a destruir.

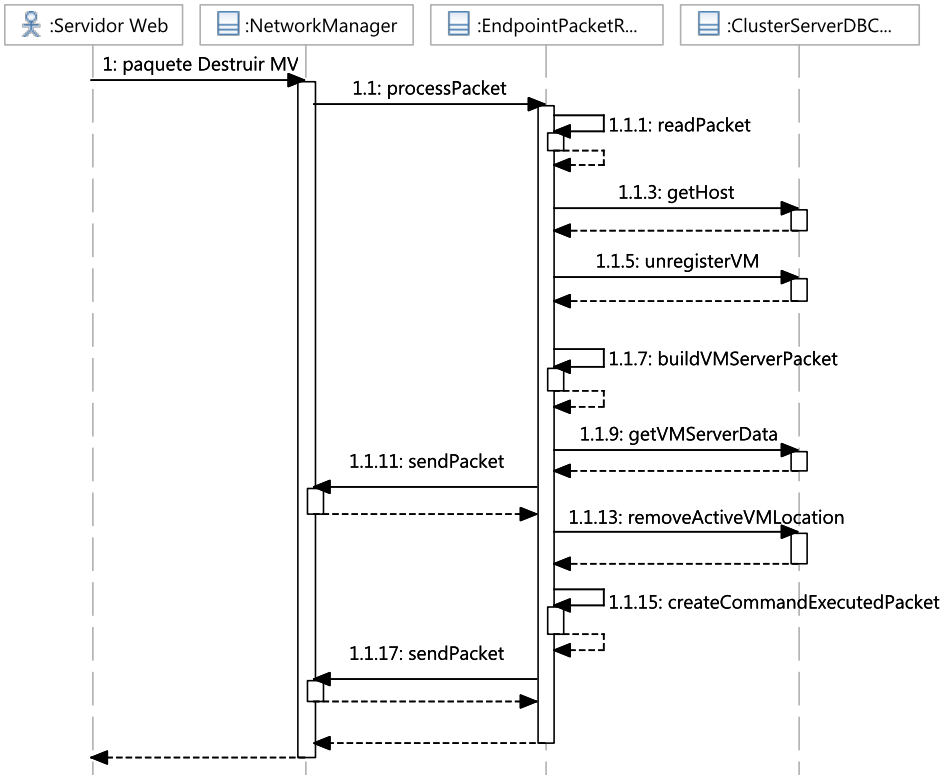

FIGURA 4.78: Apagado forzoso de una máquina virtual: secuencia básica

El diagrama de secuencia de la figura 4.78 muestra cómo se procesan estos paquetes en el servidor de *cluster*. Los pasos que se siguen son los siguientes:

1. se averigua qué servidor de máquinas virtuales alberga la máquina virtual.

   Como en la base de datos del servidor de *cluster* está almacenada la ubicación de cada máquina virtual, la petición de apagado forzoso se enviará únicamente a ese servidor de máquinas virtuales. Esto nos permite ahorrar ancho de banda.

2. se construye un paquete del tipo Apagado Forzoso MV. Acto seguido, se envía ese paquete al anfitrión de la máquina virtual.

3. se elimina la ubicación de la máquina virtual.

4. finalmente, se construye un paquete del tipo Comando ejecutado y se le envía al servidor web.

4.5.7.7.5. Apagado forzoso de una máquina virtual: tratamiento de errores

Al procesar una petición de apagado forzoso de una máquina virtual, se producirá un error si dicha máquina virtual no se encuentra registrada en el servidor de *cluster*.

Tras consultar la ubicación de la máquina virtual en la base de datos, se detectará el error y

1. se construirá un paquete del tipo Error en destrucción de máquina virtual. Estos paquetes contienen el identificador único de la petición y un código de descripción del error.

2. acto seguido, se enviará dicho paquete al servidor web.

4.5.7.7.6. Reinicio forzoso de una máquina virtual: secuencia básica

Cuando una máquina virtual se cuelga, los usuarios también podrán forzar su reinicio. En este caso, el servidor web enviará un paquete del tipo Reiniciar máquina virtual al servidor de *cluster*.

Estos paquetes se procesan casi igual que los paquetes del tipo Destrucción de máquina virtual. No obstante, existen dos diferencias:

- se envía un paquete del tipo Reinicio forzoso de máquina virtual al anfitrión de la máquina virtual.

- la máquina virtual no se borra de la base de datos ya que, cuando se reinicia, no se destruye.

4.5.7.7.7. Reinicio forzoso de una máquina virtual: tratamiento de errores

Cuando se procesa una petición de reinicio forzoso de una máquina virtual, se producirá un error si la máquina virtual que se desea reiniciar no existe.

Estos errores se procesan casi igual que los del apagado forzoso de una máquina virtual: en lugar de enviar paquetes de error del tipo Error en destrucción de máquina virtual, se enviarán paquetes de error del tipo Error en reinicio de máquina virtual.

4.5.7.8. Actualización del estado de las máquinas del *cluster*

Para poder realizar el balanceado de carga adecuadamente y comprobar si el *cluster* puede atender las peticiones de los usuarios, el servidor de *cluster* debe recopilar periódicamente el estado del resto de máquinas del *cluster*.

Para ello, se utiliza un hilo que, periódicamente,

- enviará dos paquetes, de los tipos Solicitud de estado y Solicitud de identificadores de máquinas virtuales activas, a todos los servidores de máquinas virtuales del *cluster*.

- envía un paquete del tipo Solicitud de estado al repositorio de imágenes.

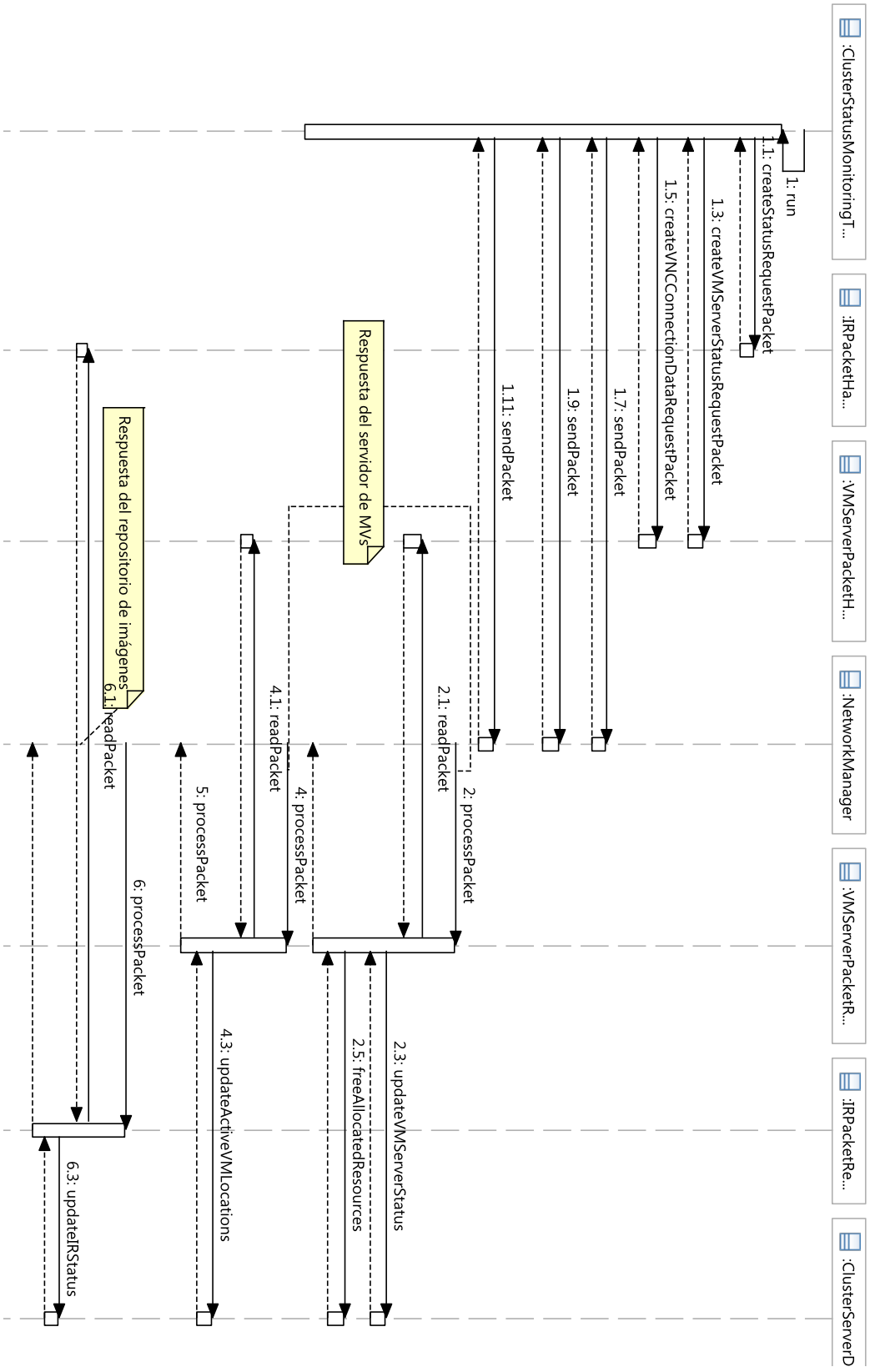

FIGURA 4.79: Actualización del estado de las máquinas del *cluster*

El intervalo de tiempo que separa los envíos de estos paquetes es configurable, y se fija en el fichero de configuración del demonio del servidor de *cluster*.

El diagrama de secuencia de la figura 4.79 muestra estos envíos de paquetes y el procesamiento de las respuestas asociadas.

A la hora de procesar las respuestas,

- cuando el servidor de *cluster* recibe un paquete del tipo Estado del servidor de máquinas virtuales,

  1. actualiza el estado del servidor de máquinas virtuales correspondiente en la base de datos del servidor de *cluster*.
  2. libera los recursos preasignados a todas las operaciones finalizadas que afectan a ese servidor de máquinas virtuales.

- cuando el servidor de *cluster* recibe un paquete del tipo Estado del repositorio, actualiza el estado del repositorio de imágenes en la base de datos del servidor de *cluster*.

- cuando el servidor de *cluster* recibe un paquete del tipo Identificadores de máquinas virtuales activas,

  1. registra la ubicación de las nuevas máquinas virtuales que residen en el servidor, y
  2. borra las ubicaciones de las máquinas virtuales que residían en el servidor y que se han apagado.

Finalmente, tras enviar los paquetes, el hilo dormirá hasta que deba realizarse la siguiente actualización del estado de las máquinas del *cluster*.

4.5.7.9. Despliegue de imágenes de disco

Los servidores de *cluster* soportan dos operaciones de despliegue de imágenes:

- **despliegue manual**. En este caso, se indica cuál es el servidor de máquinas virtuales que alojará la imagen correspondiente.

- **despliegue automático**. En este caso, sólo se solicita un número máximo de instancias al servidor de *cluster*, y este se ocupará de ubicar la imagen en tantos servidores de máquinas virtuales como sea necesario.

En esta sección, estudiaremos en detalle el funcionamiento de estas dos operaciones. Nuevamente, comenzaremos estudiando las secuencias básicas, en las que no se producen errores. Posteriormente, mostraremos cómo se detectan y tratan dichos errores.

4.5.7.9.1. Despliegue manual: secuencia básica

Cuando el servidor web envía un paquete del tipo Desplegar o borrar imagen al servidor de *cluster*, se desencadena el proceso de despliegue manual de dos imágenes de disco. Dichos paquetes contienen

- el identificador único de la petición,

- el identificador único de las imágenes de disco a desplegar,

- un *flag*, que toma el valor `False`, y

- el nombre o la dirección IP del servidor de máquinas virtuales en el que se ubicarán las imágenes de disco.

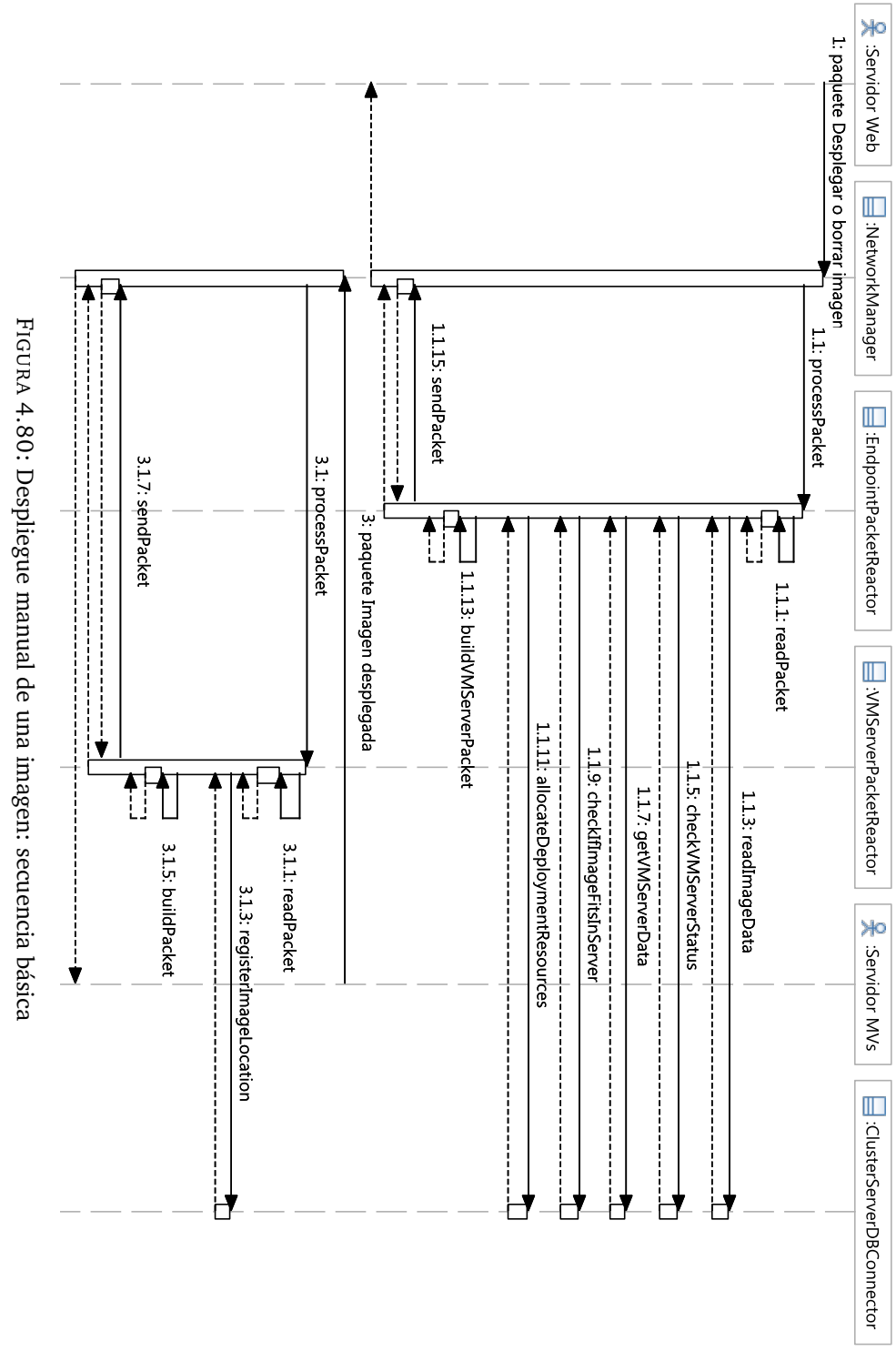

FIGURA 4.80: Despliegue manual de una imagen: secuencia básica

El diagrama de secuencia de la figura 4.80 contiene los pasos más relevantes de la operación de despliegue manual. Por claridad, al construirlo hemos supuesto que no se produce ningún error.

Cuando el servidor de *cluster* recibe el paquete, se siguen los siguientes pasos:

1. se obtienen los datos de la imagen y el estado del servidor de máquinas virtuales.
2. acto seguido, se comprueba si el servidor de máquinas virtuales está arrancado. También se comprueba si el servidor de máquinas virtuales puede albergar las instancias que utilizarán la imagen.
3. como todo es correcto, se reserva el espacio en disco que ocupará la imagen en el servidor de máquinas virtuales.
4. se envía un paquete del tipo Desplegar imagen al servidor de máquinas virtuales.
5. cuando el despliegue de la imagen finaliza, el servidor de máquinas virtuales envía al servidor de *cluster* un paquete del tipo Imagen desplegada.
6. durante el procesamiento de ese paquete, se registra la nueva ubicación de la imagen que se acaba de desplegar. Los recursos preasignados a la operación de despliegue se liberarán durante la próxima actualización del estado del servidor de máquinas virtuales.
7. finalmente, se envía un paquete del tipo Comando ejecutado al servidor web. Este contiene el identificador único de la petición.

4.5.7.9.2. Despliegue manual: tratamiento de errores

Durante el despliegue manual de una imagen, pueden producirse los siguientes errores:

- el despliegue de la imagen falla en el servidor de máquinas virtuales. Esto sólo ocurrirá si la imagen se ha borrado mientras se despliega o si se ha perdido la conexión con el repositorio de imágenes con anterioridad o durante la transferencia.
- el servidor de máquinas virtuales en el que se pretende desplegar la imagen no existe o no está arrancado.
- se ha perdido la conexión con el servidor de máquinas virtuales.
- el servidor de máquinas virtuales escogido no puede albergar la imagen.

Con independencia del error que se produzca, siempre se actúa igual:

1. si es necesario, se liberan los recursos preasignados a la operación de despliegue.
2. el servidor de *cluster* genera un paquete del tipo Error de despliegue. Estos paquetes contienen el identificador único de la petición y un código de error.
3. acto seguido, el servidor de *cluster* envía ese paquete al servidor web.

4.5.7.9.3. Despliegue automático: secuencia básica

El proceso de despliegue automático de una imagen se iniciará siempre que el servidor web envíe un paquete del tipo Despliegue automático al servidor de *cluster*. Estos paquetes contienen

- el identificador único de la petición,
- el identificador único de la imagen a desplegar, y
- el número de nuevas instancias que van a utilizar dicha imagen.

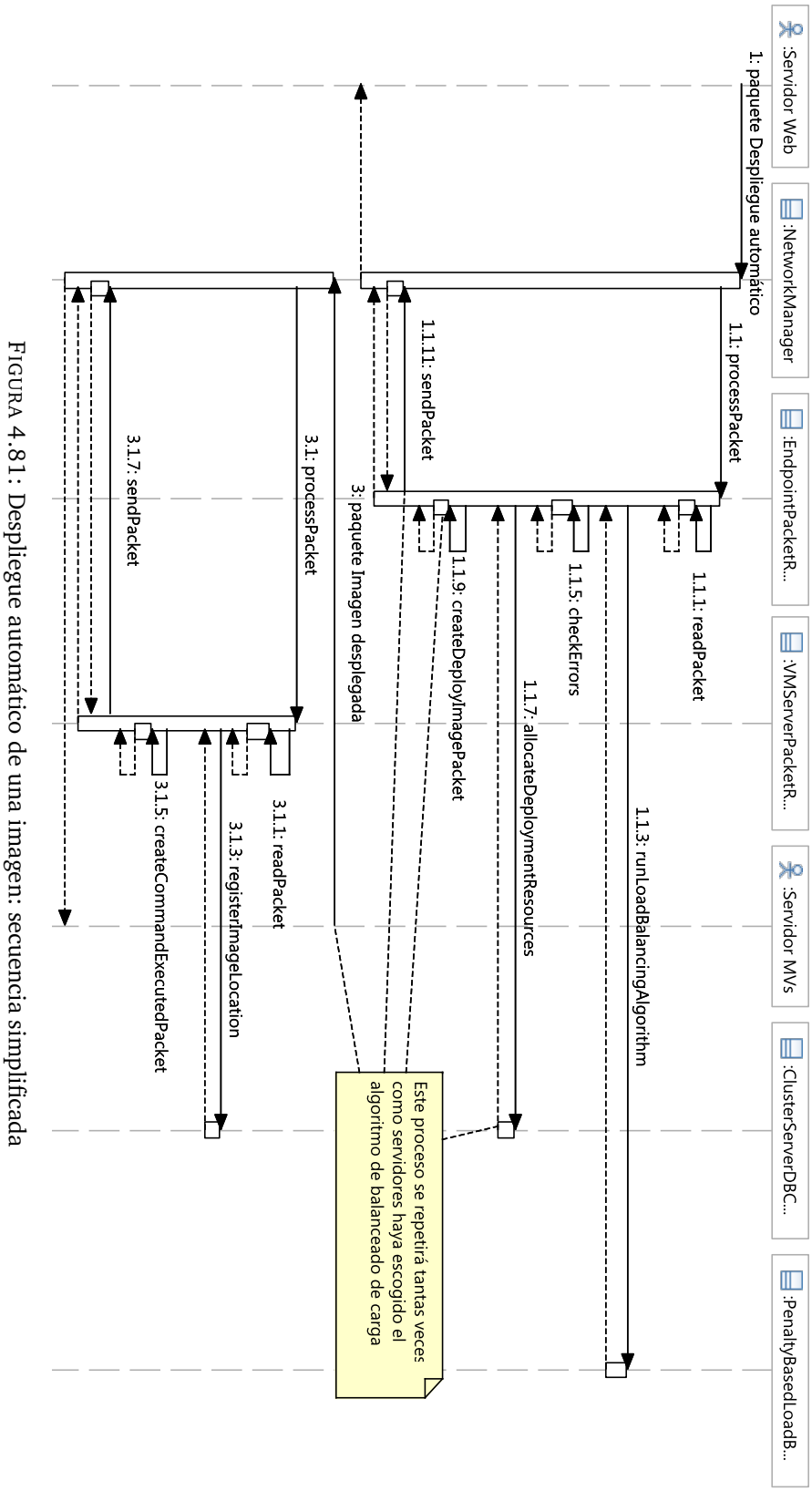

FIGURA 4.81: Despliegue automático de una imagen: secuencia simplificada

El diagrama de secuencia de la figura 4.81 muestra cómo se procesan las peticiones de despliegue automático en el servidor de *cluster*. Por claridad, a la hora de construir ese diagrama, hemos supuesto que la imagen sólo se despliega en un único servidor de máquinas virtuales, que es capaz de albergar el número de instancias solicitado.

Como podemos observar en dicho diagrama, se siguen estos pasos:

1. tras recibir el paquete del tipo Despliegue automático, se ejecuta el algoritmo de balanceado de carga.
2. acto seguido, se comprueba si los servidores seleccionados pueden albergar el número de instancias solicitado.
3. para cada servidor de máquinas virtuales seleccionado por el algoritmo,
   a) se reservan los recursos requeridos por el proceso de despliegue,
   b) se construye y envía un paquete del tipo Desplegar imagen a dicho servidor.
4. a medida que se reciben los paquetes del tipo Imagen desplegada, se registran las nuevas ubicaciones de las imágenes de disco en la base de datos del servidor de *cluster*.
5. cuando se recibe el último paquete del tipo Imagen desplegada, se construye un paquete del tipo Comando ejecutado y se envía al servidor web.

4.5.7.9.4. Despliegue automático de imágenes de disco: tratamiento de errores

Durante el proceso de despliegue automático de imágenes de disco, se producirán errores si

- las imágenes de disco de la petición no existen.
- el número máximo de instancias que es posible obtener es inferior al dado.
- el proceso de despliegue falla en algún servidor de máquinas virtuales.

Los dos primeros errores se detectarán cuando el algoritmo de balanceado de carga termina de ejecutarse. En estos casos, el servidor de *cluster* abortará la operación y enviará al servidor web un paquete del tipo Error de despliegue automático, que contiene el identificador único de la petición y un código de error. Este proceso se refleja en el diagrama de secuencia de la figura 4.82.

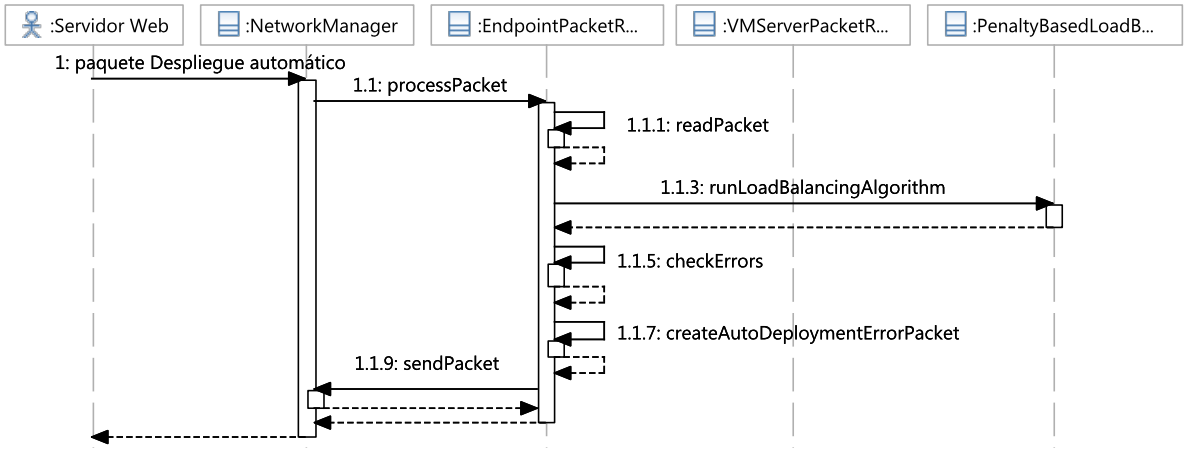

FIGURA 4.82: Despliegue automático de imágenes de disco: tratamiento del primer tipo de error

Por otra parte, el despliegue automático de imágenes de disco también fallará cuando se produzca un error de despliegue en un servidor de máquinas virtuales. Por claridad, mostraremos cómo se actúa en estos casos mediante un ejemplo.

Supongamos que el algoritmo de balanceado de carga escoge los servidores de máquinas virtuales A, B y C para realizar la petición de despliegue. Supongamos también que la operación de despliegue falla en los servidores A y B, y que tiene éxito en el servidor C.

El sistema actuará como se indica en el diagrama de secuencia de la figura 4.83. En él hemos omitido la fase inicial de la operación de despliegue, que es idéntica a la que presentamos en la sección 4.5.7.9.3.

Como podemos observar en el diagrama de secuencia, cuando se recibe el primer paquete del tipo Error de despliegue,

1. se construye un paquete del tipo Error de despliegue automático, y se envía al servidor web.
2. cuando se reciban los demás paquetes asociados a la operación, se procederá de la siguiente manera:
   - si se recibe un paquete del tipo Error de despliegue, una operación de despliegue más habrá fallado. En estos casos, se liberarán los recursos preasignados a la operación de despliegue y se ignorará el error.
   - si se recibe un paquete del tipo Imagen desplegada, se registrará la ubicación de la nueva imagen. En la próxima actualización del estado del *cluster*, se liberarán los recursos asignados a la operación.

Es importante notar que los errores del tercer tipo se producen únicamente en circunstancias excepcionales. Además, para subsanarlos resulta imprescindible la intervención de un administrador, ya que en estos casos al menos una de las máquinas del *cluster* está funcionando incorrectamente.

4.5.7.10. Borrado de imágenes de disco

Los servidores de *cluster* también soportan dos operaciones de borrado de imágenes:

- **borrado manual**. Esta operación borra la copia de una imagen almacenada en cierto servidor de máquinas virtuales.
- **borrado total**. Esta operación borra todas las copias de una imagen almacenadas en todos los servidores de máquinas virtuales, y también el fichero comprimido asociado a la misma, que se almacena en el repositorio de imágenes.

En esta sección, estudiaremos estas dos operaciones en detalle. Nuevamente, comenzaremos estudiando las secuencias básicas, en las que no se producen errores. Posteriormente, mostraremos cómo se detectan y tratan dichos errores.

4.5.7.10.1. Borrado manual de imágenes de disco: secuencia básica

El servidor de *cluster* realizará una operación de borrado manual tras recibir un paquete del tipo Desplegar o borrar imagen. Este paquete contiene

- el identificador único de la petición,
- el identificador único de las imágenes de disco a desplegar,
- un *flag*, que toma el valor `False`,
- el nombre o la dirección IP del servidor de máquinas virtuales en el que se ubicarán las imágenes de disco.

Durante el procesamiento de estos paquetes, se siguen, esencialmente, los mismos pasos que en el despliegue manual de una imagen. No obstante, existen tres diferencias:

- en lugar de comprobar si el servidor de máquinas virtuales puede albergar las instancias de la imagen imagen, se comprueba si este contiene la imagen.

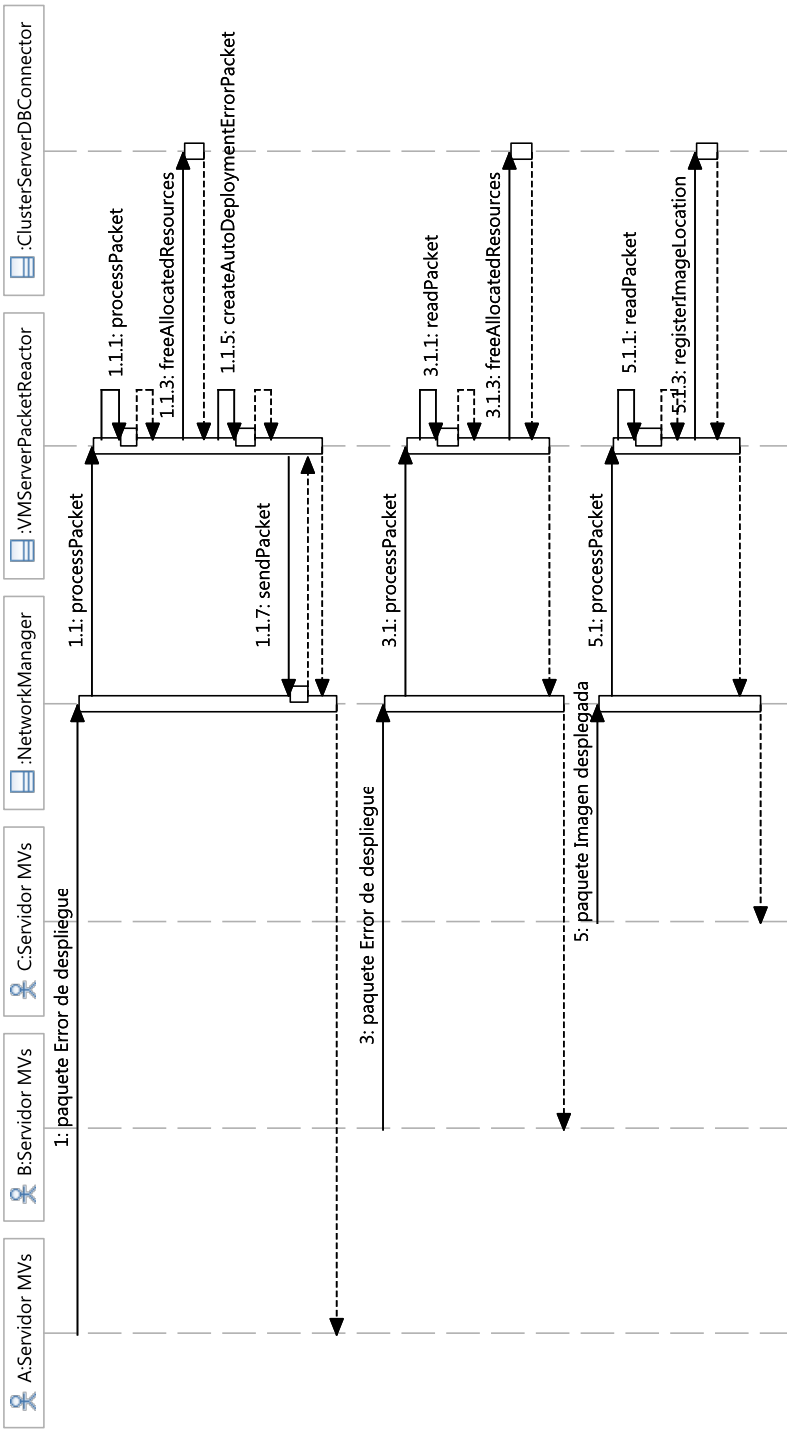

FIGURA 4.83: Despliegue automático de imágenes de disco: tratamiento del segundo tipo de error

- para borrar la imagen del servidor de máquinas virtuales, se le envía un paquete del tipo Borrar imagen.

- cuando el servidor de máquinas virtuales borra la imagen, este envía un paquete del tipo Imagen borrada al servidor de *cluster*. Sólo en este momento se borrará la ubicación correspondiente de las imágenes de disco.

Estos pasos aparecen en el diagrama de secuencia de la figura 4.84.

Es importante notar que el *flag* de los paquetes del tipo Desplegar o borrar imagen nos permite distinguir las operaciones de despliegue de las operaciones de borrado manual de imágenes, ya que ambas utilizan el mismo tipo de paquete.

4.5.7.10.2. Borrado manual de imágenes de disco: secuencia básica

Durante el borrado manual de imágenes de disco, se producirán errores cuando

- el servidor de máquinas virtuales afectado por la petición está apagado, o cuando

- el servidor de máquinas virtuales no contiene las imágenes de disco.

En ambos casos, se procede de la misma manera:

1. se aborta el proceso de borrado.

2. se envía un paquete del tipo Error al borrar imagen al servidor web.

4.5.7.10.3. Borrado total de una imagen: secuencia básica

Cuando el servidor de *cluster* recibe un paquete del tipo Borrar imagen de infraestructura, se inicia el proceso de borrado total de la imagen. Como ya hemos dicho, en este caso

- se borrarán todas las copias de la imagen existentes en todos los servidores de máquinas virtuales, con independencia de su estado, y

- se borrará el fichero comprimido del repositorio de imágenes.

Por claridad, mostraremos el funcionamiento de esta operación utilizando un ejemplo. Supongamos que

- se desea borrar la imagen con identificador 1.

- esa imagen se encuentra desplegada en los servidores de máquinas virtuales A y B. El servidor A está arrancado, pero el servidor B, no.

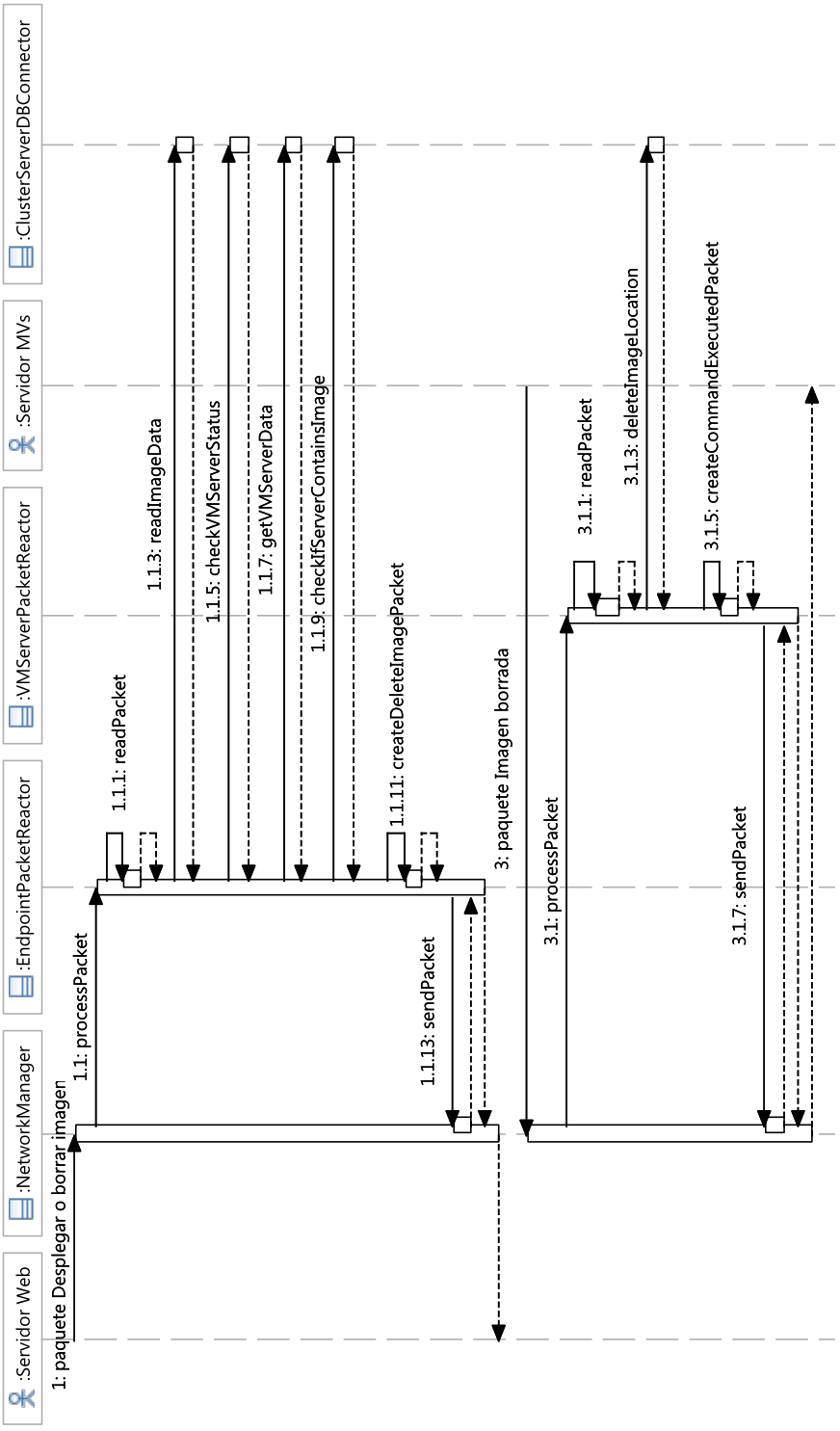

FIGURA 4.84: Borrado manual de una imagen: secuencia básica

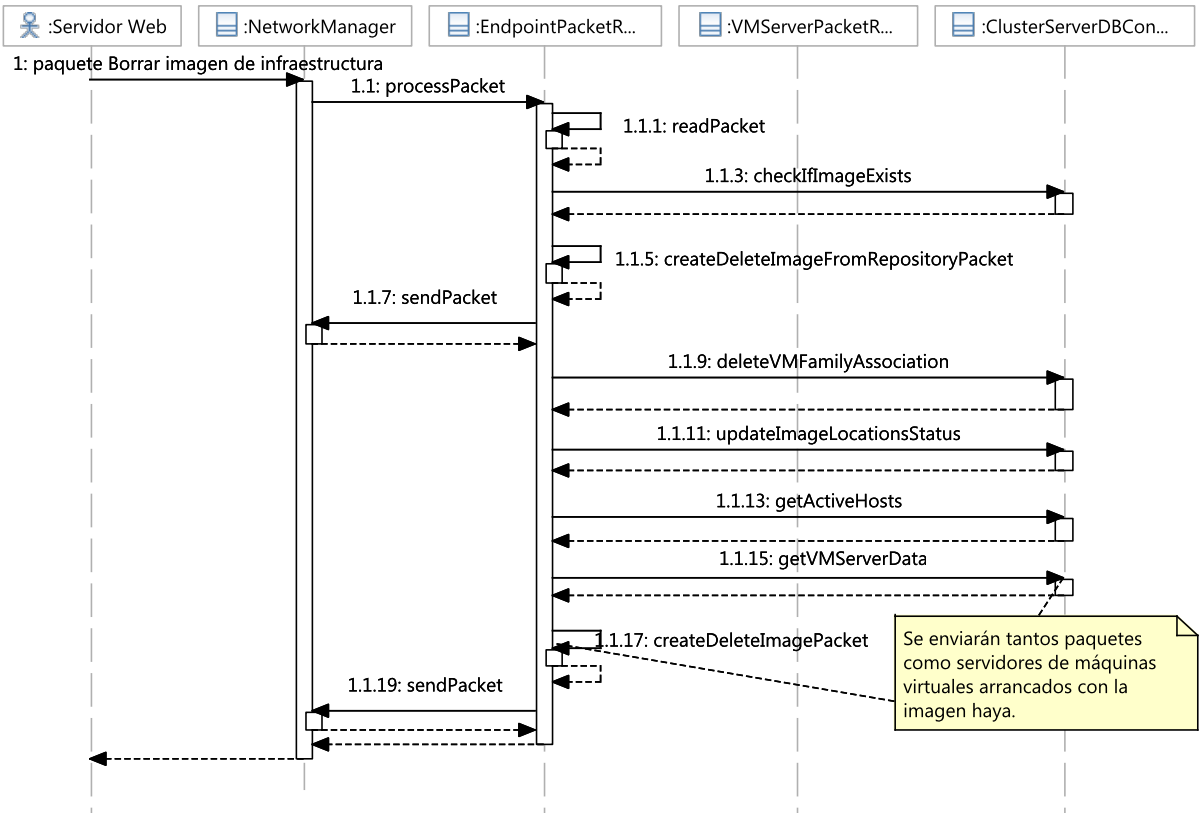

FIGURA 4.85: Borrado total de una imagen: fase inicial

El diagrama de secuencia de la figura 4.85 muestra la primera fase de la operación de borrado total. En ella, se siguen los siguientes pasos:

1. se solicita el borrado de la imagen al repositorio de imágenes.
2. se borra la asociación entre la imagen y una familia de máquinas virtuales.
3. se modifica el estado de todas las copias existentes de la imagen. A partir de este momento, ninguna de ellas podrá utilizarse para crear una nueva máquina virtual.
4. se envía un paquete del tipo Borrar imagen a todos los servidores de máquinas virtuales arrancados que tienen una copia de la imagen.

En la segunda fase de la operación, se procesan las respuestas del repositorio de imágenes y del servidor de máquinas virtuales. Dicho procesamiento se recoge en el diagrama de secuencia de la figura 4.86.

Como podemos observar en el diagrama de secuencia,

- el paquete del tipo Solicitud de borrado recibida enviado por el repositorio de imágenes se descarta.
- cuando el servidor de máquinas virtuales A envía el paquete Imagen borrada al servidor de *cluster*, este
  - borra la ubicación de la copia de la imagen.
  - comprueba si la imagen está ubicada en algún servidor de máquinas virtuales más. Como el servidor B está apagado, su copia no se ha borrado. Por ello, la operación de borrado total aún no ha finalizado.

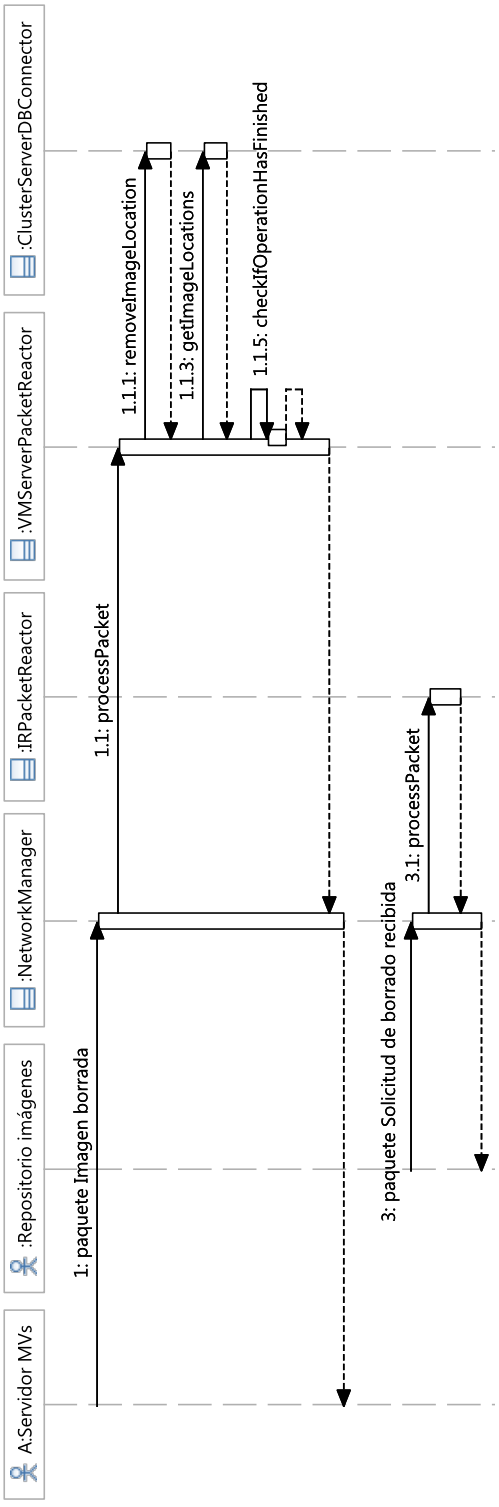

FIGURA 4.86: Borrado total de una imagen: procesamiento de los paquetes de confirmación

Debemos recordar que los paquetes del tipo Imagen borrada contienen el identificador único de la petición y el identificador único de la imagen a borrar. Esto permite al servidor de *cluster* identificar la petición de borrado.

Por otra parte, como dijimos al presentar el proceso de arranque de un servidor de máquinas virtuales, cuando se arranque el servidor de máquinas virtuales B se le enviará un paquete del tipo Borrar imagen para borrar en él la imagen 1.

La respuesta del servidor de máquinas virtuales se procesará casi igual que en el diagrama de secuencia de la figura 4.86.

No obstante, tras procesar el paquete del tipo Imagen borrada no habrá ninguna otra copia de la imagen 1, por lo que construirá un paquete del tipo Comando ejecutado y se enviará al servidor web.

4.5.7.11. Borrado total de una imagen: tratamiento de errores

Mientras se procesa una petición de borrado total, se producirán errores cuando

- la imagen a borrar no existe.
- el borrado de la imagen falla en algún servidor de máquinas virtuales.

Los errores del primer tipo se detectan tan pronto como empieza a procesarse la petición. En estos casos,

1. se aborta el procesamiento de la petición.
2. se construye un paquete del tipo Error al borrar imagen de infraestructura. Estos paquetes contienen el identificador único de la imagen a borrar y un código de descripción del error.
3. se envía dicho paquete al servidor web.

Por otra parte, los errores del segundo tipo se tratan igual que los errores de despliegue automático:

1. cuando se recibe el primer paquete del tipo Error al borrar imagen procedente de un servidor de máquinas virtuales,

   a) no se borra la ubicación de la imagen de la base de datos.

   b) se construye un paquete del tipo Error al borrar imagen de infraestructura y se envía al servidor de *cluster*.

   Debemos recordar que los paquetes del tipo Error al borrar imagen contienen el identificador de la petición y el identificador de la imagen a borrar.

2. si más servidores de máquinas virtuales envían un paquete del tipo Error al borrar imagen al servidor de *cluster*, estos se ignorarán.
3. si un servidor de máquinas virtuales envía un paquete del tipo Imagen borrada al servidor de *cluster*, se borrará la ubicación de la imagen correspondiente.

El diagrama de secuencia que muestra estas interacciones es prácticamente idéntico al de la figura 4.83. Por ello, hemos decidido no incluirlo.

4.5.7.12. Creación y edición de imágenes

Desde el punto de vista del servidor de *cluster*, los procesos de creación y edición de imágenes se dividen en dos fases:

- el arranque de la máquina virtual que se usará para fijar su contenido, y
- en el caso de las operaciones de edición de imágenes, la actualización de todas las copias existentes de la imagen. Esto es fundamental para que se apliquen los cambios realizados en la imagen.

Para empezar, nos centraremos en la primera fase. Por claridad, en todos los casos siempre empezaremos mostrando las secuencias básicas, en las que no se producen errores. Posteriormente, mostraremos las secuencias alternativas, que se corresponden con el tratamiento de errores.

4.5.7.12.1. Arranque de la máquina virtual: flujo básico

Puesto que el servidor de *cluster* no interactúa con el repositorio de imágenes durante los procesos de creación y edición de imágenes de disco, la primera fase de ambos es casi idéntica.

La creación de nuevas imágenes comienza con la recepción de un paquete del tipo Crear imagen en el servidor de *cluster*. Asimismo, la edición de imágenes comienza siempre con la recepción de un paquete del tipo Editar imagen en el servidor de *cluster*.

Los paquetes del tipo Crear imagen contienen

- el identificador de la imagen que se usará como base para crear la nueva imagen. Las imágenes pueden crearse a partir de cualquier imagen existente, con independencia de que esta sea una imagen base o no.
- el identificador de la petición.
- el identificador del usuario que ha enviado la petición.

Por otra parte, los paquetes del tipo Editar imagen contienen

- el identificador de la imagen que se pretende editar.
- el identificador de la petición.
- el identificador del usuario que ha enviado la petición.

El diagrama de secuencia de la figura 4.87 muestra cómo se procesa un paquete del tipo Editar imagen. Como podemos observar en él, se dan los siguientes pasos:

1. se comprueba que la imagen no está afectada por otra petición de edición.
2. se ejecuta el algoritmo de balanceado de carga para averiguar a qué servidor de máquinas virtuales se enviará la petición.
3. se bloquea la imagen. Hasta que este proceso de edición termine, ningún otro usuario podrá editarla.
4. se reservan recursos en el servidor de máquinas virtuales escogido y en el repositorio de imágenes. Esto último resulta imprescindible, ya que la imagen modificada puede ocupar en el repositorio de imágenes más espacio que la imagen original.
5. se crea un paquete del tipo Editar imagen, que tendrá el *flag* activado. Acto seguido, se enviará dicho paquete al servidor de máquinas virtuales que ha escogido el servidor de *cluster*.
6. el servidor de máquinas virtuales descarga la imagen y crea la máquina virtual que se usará para editarla. Acto seguido, envía al servidor de *cluster* los datos de conexión del servidor VNC asociado a esa máquina virtual.

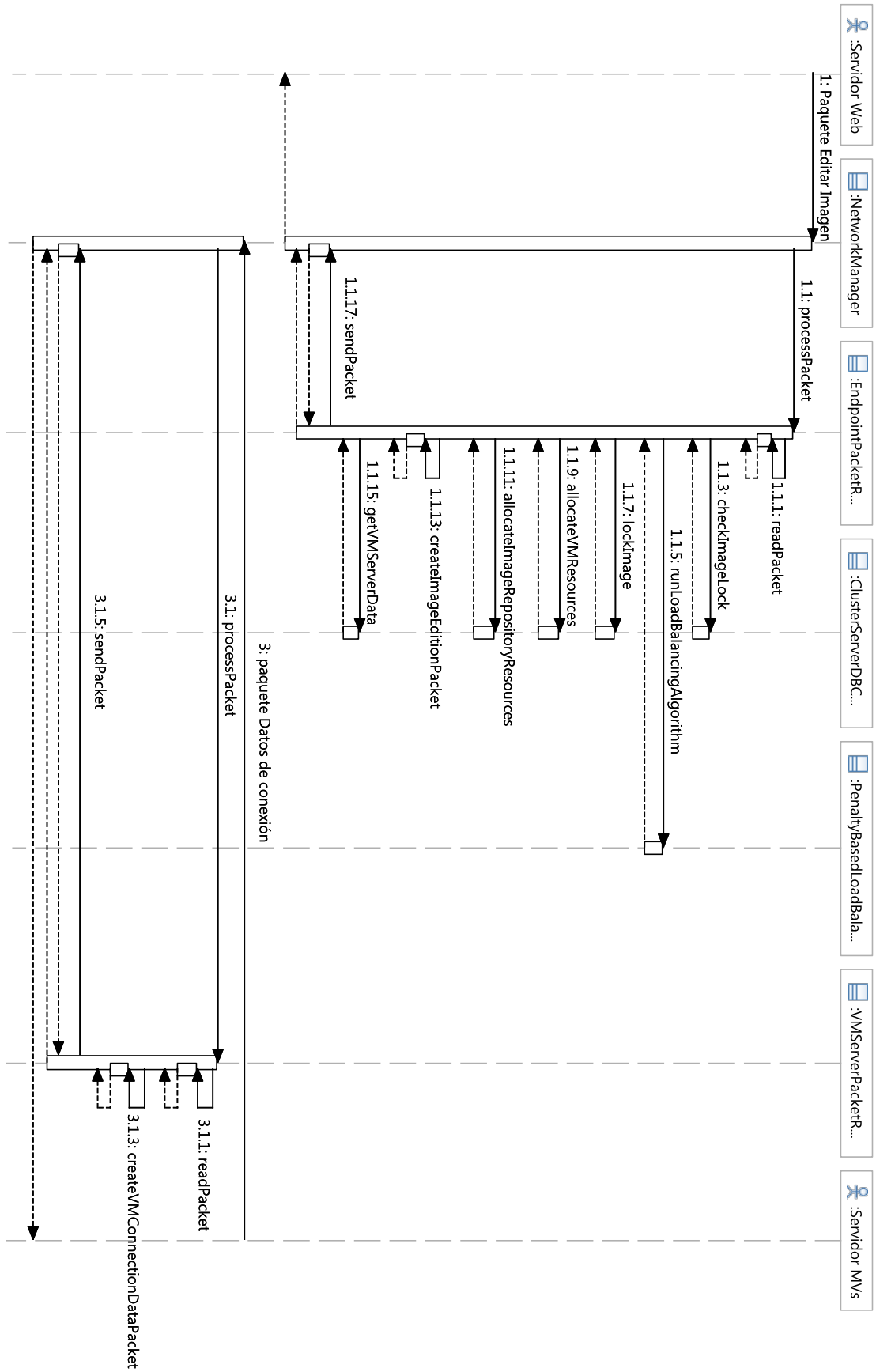

FIGURA 4.87: Edición de imágenes de disco: arranque de la máquina virtual sin errores

7. el servidor de *cluster* envía esta información al servidor web.

Los paquetes del tipo Crear imagen se procesan de forma muy parecida. No obstante, hay tres diferencias:

- el paquete Crear imagen que se envía al servidor de máquinas virtuales tiene el *flag* desactivado.
- a partir del identificador único de la petición, se crea un identificador temporal de la imagen. Asimismo, se asocia dicho identificador temporal con una familia de máquinas virtuales.
- como la imagen que se descarga del repositorio no se modifica, no es necesario bloquearla.

Finalmente, debemos recordar que no todos los servidores de máquinas virtuales podrán utilizarse para crear o editar imágenes de disco: el servidor de *cluster* sólo considerará aquellos que tengan el *flag* de edición activado.

4.5.7.12.2. Arranque de la máquina virtual: tratamiento de errores

Al arrancar una máquina virtual que se utiliza para crear o editar imágenes, se producirán errores si

- no existe ningún servidor de edición que pueda albergar la máquina.
- no existe suficiente espacio en disco en el repositorio de imágenes.
- no es posible descargar el fichero comprimido con las imágenes de disco desde el repositorio de imágenes.
- el proceso de arranque de la máquina virtual falla.

Asimismo, cuando se edita una imagen también se producirá un error cuando otro usuario ya está editándola.
Sea cual sea el error que se produzca, siempre se procede de forma similar:

1. se aborta el proceso de creación o edición de la imagen
2. si es necesario, se liberan todos los recursos asignados a la máquina virtual y se desbloquea la imagen
3. si la petición que ha fallado es una petición de creación de imagen, se crea un paquete del tipo Error de creación de imagen; si es una petición de edición de imagen, se crea una petición del tipo Error de edición de imagen.

   Ambos tipos de paquete contienen el identificador único de la petición y un código de descripción del error.
4. se envía dicho paquete al servidor web.

4.5.7.12.3. Reserva de recursos

Al crear o editar imágenes de disco, se reservan, entre otras cosas,

- el espacio en disco que ocupará el fichero comprimido con los nuevos datos en el repositorio de imágenes, y
- el espacio en disco que ocupará la imagen en el servidor de máquinas virtuales.

Como vimos en la sección 4.79, al recibir un paquete con información de estado procedente del servidor de máquinas virtuales o del repositorio de imágenes, se liberarán los recursos preasignados en dicha máquina.

No obstante, esto no sucede exactamente así cuando se crean o editan imágenes. Cuando se apaga la máquina virtual que se usa para editar la imagen, es necesario,

- volver a comprimir las imágenes de disco, lo cual requiere espacio en disco en el servidor de máquinas virtuales, y
- subir un fichero comprimido al repositorio de imágenes, cosa que también requiere espacio en disco en el repositorio de imágenes.

Por ello, tras recibir los paquetes de estado enviados por el repositorio de imágenes y el servidor de máquinas virtuales, el servidor de *cluster* mantendrá reservado el espacio en disco que ocupará fichero comprimido en estas máquinas. Este espacio se liberará cuando finalice el proceso de creación o edición de la imagen.

4.5.7.12.4. Apagado de la máquina virtual

Como vimos en la sección 4.5.6.9, cuando se apaga una máquina virtual que se utiliza para crear o editar imágenes, estas se transfiere un fichero comprimido al repositorio de imágenes.

Cuando finaliza esa transferencia, el servidor de máquinas virtuales enviará un paquete del tipo Imagen editada al servidor de *cluster*. Dicho paquete contiene el identificador único de la petición y el identificador único de la máquina virtual.

El diagrama de secuencia de la figura 4.88 muestra cómo se procesan los paquetes del tipo Imagen editada en el servidor de *cluster* asociados a una petición de creación de una imagen. Estos paquetes contienen el identificador de la nueva imagen y el identificador de la petición.

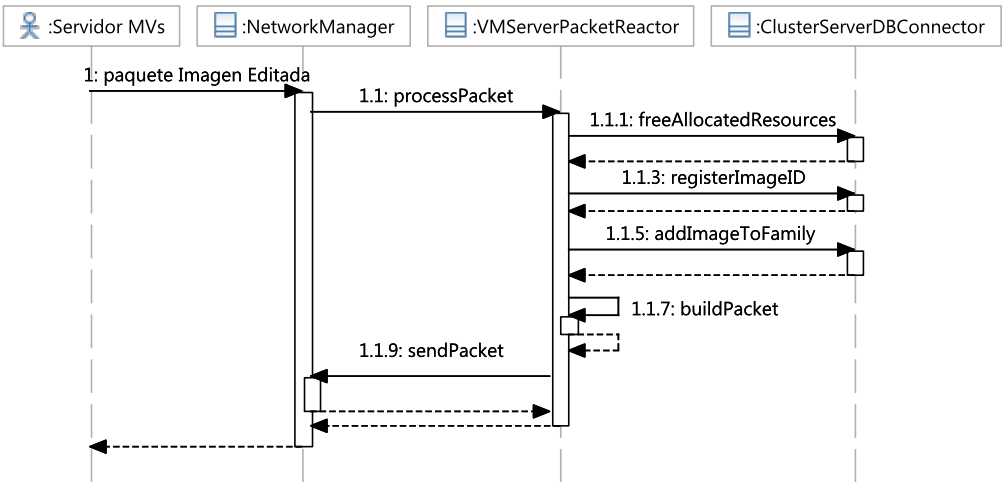

FIGURA 4.88: Creación de imágenes de disco: apagado de la máquina virtual

Como podemos observar en el ese diagrama, al procesar el paquete del tipo Imagen editada,

1. se inicia la liberación de los recursos asociados a la petición. Estos se liberarán definitivamente cuando se realice la siguiente actualización del estado del *cluster*.
2. se registra el identificador definitivo de la nueva imagen, así como la familia de máquinas virtuales a la que está asociado.
3. se construye un paquete del tipo Imagen creada. Estos paquetes contienen el identificador único de la petición y el identificador de las nuevas imágenes.

4. se envía dicho paquete al servidor web.

A partir de este momento, será posible desplegar las nueva imagen o continuar editando su contenido.

Por otra parte, si el paquete del tipo Imagen editada se corresponde con una petición de imágenes, se procede de forma muy similar al caso anterior. No obstante, hay una diferencia: no es necesario registrar el identificador de la imagen ni la familia de máquinas virtuales a la que está asociado.

Finalmente, a partir de este momento será posible

- seguir editando el contenido de la imagen de disco, o
- actualizar todas sus copias.

Estudiaremos este último proceso en la sección 4.5.7.12.6.

4.5.7.12.5. Apagado de la máquina virtual: tratamiento de errores

Cuando la máquina virtual se apaga, pueden producirse errores al subir el fichero comprimido al repositorio de imágenes.

En estos casos, el servidor de máquinas virtuales enviará un paquete del tipo Error de edición de imagen al servidor de *cluster*. Estos paquetes se procesan como vimos en la sección 4.5.7.12.2.

4.5.7.12.6. Actualización de las copias de una imagen

Para actualizar todas las copias de una imagen que ha terminado de editarse, el servidor web enviará al servidor de *cluster* un paquete del tipo Despliegue automático. Estos paquetes contienen

- el identificador de la imagen a desplegar,
- un número de instancias, que en este caso es negativo, y
- el identificador único de la petición.

Estos paquetes se procesan de forma muy parecida a los paquetes del tipo Borrar imagen de infraestructura. No obstante, existen tres diferencias:

- la imagen no se borra del repositorio.
- los paquetes que se envían a los servidores de máquinas virtuales son del tipo Desplegar imagen.
- cuando se produce un error, se envía un paquete del tipo Error de despliegue automático al servidor de *cluster*.

El lector podrá encontrar la explicación del procesamiento de los paquetes del tipo Borrar imagen de infraestructura en las secciones 4.5.7.10.3 y 4.5.7.11.

4.5.7.13. Solicitudes de estado

Para mostrar información actualizada a los profesores y administradores, el servidor web debe averiguar periódicamente

- el estado de los servidores de máquinas virtuales arrancados,
- el estado del repositorio de imágenes, y
- la distribución de las copias de las imágenes.

Por otra parte, para que todos los usuarios puedan reconectarse a los servidores VNC de las máquinas virtuales, el servidor web también debe conocer los correspondientes datos de conexión.

Para que esto sea posible, el servidor web enviará periódicamente al servidor de *cluster* cinco peticiones, utilizando para ello paquetes de los tipos Solicitud de estado de servidores de máquinas virtuales, Solicitud de estadísticas servidores de máquinas virtuales, Solicitud de estado del repositorio, Solicitud de distribución de imágenes y Solicitud de datos de conexión VNC. Todos los paquetes de estos tipos no contienen información adicional.

Para procesar los paquetes de los cuatro primeros tipos, el servidor de *cluster*

1. realiza la consulta en la base de datos.

2. escribe la salida de la consulta en paquetes de los tipos Estado de servidores de máquinas virtuales, Estado del repositorio y Distribución de imágenes respectivamente.

Finalmente, para procesar los paquetes del tipo Solicitud de datos de conexión VNC, el servidor de *cluster*

1. envía un paquete del tipo Solicitud de datos de conexión VNC al servidor de máquinas virtuales.

2. cuando recibe el paquete Datos de conexión VNC procedente del servidor de máquinas virtuales, lo reenvía al servidor web.

4.5.7.14. Apagado de todas las máquinas del *cluster*

Cuando el servidor web envía un paquete del tipo Apagar infraestructura al servidor de *cluster*, este apagará todas las máquinas del *cluster* y terminará. El diagrama de secuencia de la figura 4.89 muestra el proceso de apagado de todas las máquinas del *cluster*. En él, es posible distinguir los siguientes pasos:

1. se determina qué servidores de máquinas virtuales están arrancados.

2. se construye un paquete del tipo Apagado con espera. Acto seguido, se enviará a todos los servidores de máquinas virtuales arrancados.

   A medida que los servidores de máquinas virtuales reciban el paquete, irán apagándose progresivamente.

3. se construye un paquete del tipo Apagado. A continuación, se envía al repositorio de imágenes.

   Cuando este reciba el paquete, se apagará.

4. el hilo principal deja inmediatamente de monitorizar los comandos de arranque de máquinas virtuales.

5. el hilo principal detiene el hilo de actualización del estado del *cluster*.

6. el hilo principal cierra todas las conexiones de red. A partir de este momento, el servidor de *cluster* dejará de atender peticiones.

Es importante notar que, como dijimos en la sección 4.5.5.7, el servicio de red debe detenerse desde el hilo principal. Si no lo hacemos así, se producirá un cuelgue.

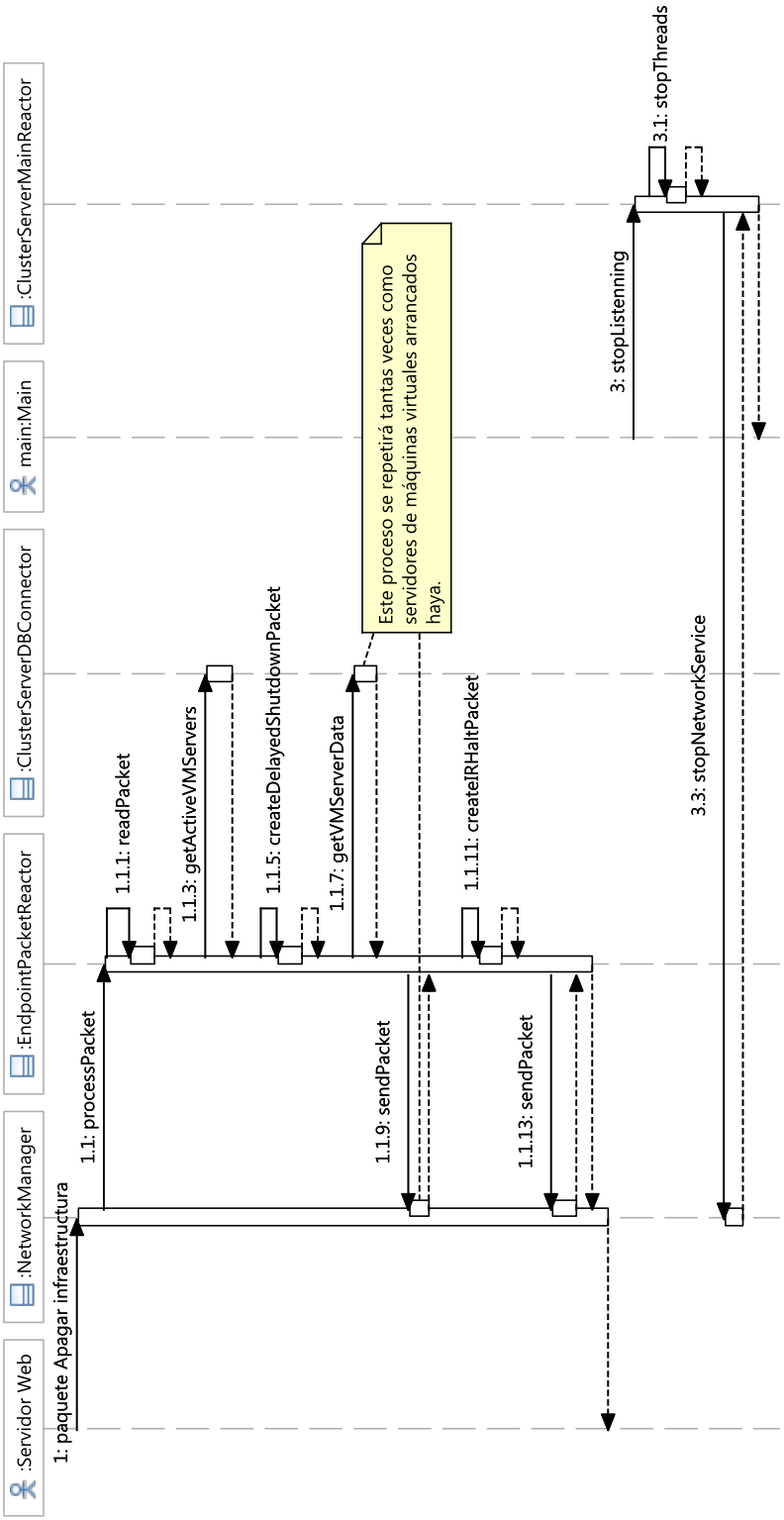

FIGURA 4.89: Apagado de todas las máquinas del *cluster*

4.5.7.15. Formatos de paquete

En esta sección, mostraremos detalladamente las características de los tipos de paquete asociados al subsistema servidor de *cluster*. Los cuadros 4.8 y 4.9 muestran los códigos, prioridades y las constantes del tipo enumerado `CLUSTER_SERVER_PACKET_T` (definido en el paquete `clusterServer.packetHandling`) asociados a cada clase de paquete.

Como dijimos en la sección 4.5.3.3, cuanto menor es la valor numérico de la prioridad de un paquete, más prioritario es.

El contenido de los distintos tipos de paquetes es el siguiente:

- los paquetes del tipo **Registro de servidor de máquinas virtuales** y **Cambiar configuración de servidor de máquinas virtuales** contienen un nombre, una dirección IP, el identificador de la petición, un *flag* y un puerto.
- los paquetes de los tipos **Solicitud de estado de servidores de máquinas virtuales**, **Solicitud de distribución de imágenes**, **Solicitud de datos de conexión VNC**, **Apagar infraestructura**, **Solicitud estado del repositorio** y **Solicitud de estadísticas de servidores de máquinas virtuales** no contienen información adicional.
- los paquetes del tipo **Estado de servidores de máquinas virtuales** contienen las direcciones IP, el número de CPUs físicas, el número de CPUs virtuales, la memoria RAM utilizada, la memoria RAM instalada, el espacio de almacenamiento utilizado, el espacio de almacenamiento total, el espacio de almacenamiento temporal utilizado y el espacio de almacenamiento temporal total de los servidores de máquinas virtuales arrancados.
- los paquetes del tipo **Distribución de imágenes** contienen los nombres de todos los servidores de máquinas virtuales y los identificadores de las imágenes que estos albergan.
- los paquetes de los tipos **Borrar o dar de baja servidor de máquinas virtuales** y **Arrancar servidor de máquinas virtuales** contienen el identificador único de la petición y el nombre o la dirección IP de un servidor de máquinas virtuales.
- los paquetes del tipo **Arranque de máquina virtual** contienen un identificador de usuario, el identificador de la imagen a arrancar y el identificador de la petición.
- los paquetes del tipo **Datos de conexión a máquina virtual** contienen la dirección IP, el puerto y la contraseña del servidor VNC asociado a una máquina virtual.
- los paquetes del tipo **Comando ejecutado** contienen el identificador de una petición.
- los paquetes **Apagado forzoso de máquina virtual** y **Reinicio forzoso de máquina virtual** contienen un identificador de petición y un identificador de máquina virtual.
- los paquetes del tipo **Estado del repositorio** contienen el espacio en disco usado y disponible en el repositorio de imágenes.
- los paquetes del tipo **Desplegar imagen** contienen un identificador de imagen y un identificador de petición.
- los paquetes del tipo **Borrar imagen** contienen un nombre, un identificador de petición y un identificador de imagen.
- los paquetes de los tipos **Crear imagen** y **Editar imagen** contienen un identificador de imagen, un identificador de petición y un identificador de usuario.
- los paquetes de los tipos **Imagen creada** y **Imagen editada** contienen un identificador de imagen y un identificador de petición.

Tipo de paquete	Código	Prioridad	Constante (tipo enumerado)
Registro de servidor de máquinas virtuales	1	4	REGISTER_VM_SERVER
Error en el registro de servidor de máquinas virtuales	2	3	VM_SERVER_REGISTRATION_ERROR
Solicitud de estado de servidores de máquinas virtuales	3	10	QUERY_VM_SERVERS_STATUS
Estado de servidores de máquinas virtuales	4	10	VM_SERVERS_STATUS_DATA
Solicitud de distribución de imágenes	5	10	QUERY_VM_DISTRIBUTION
Distribución de imágenes	6	10	VM_DISTRIBUTION_DATA
Borrar o dar de baja servidor de máquinas virtuales	7	4	UNREGISTER_OR_SHUTDOWN _VM_SERVER
Arranque de servidor de máquinas virtuales	8	4	BOOTUP_VM_SERVER
Error de arranque de servidor de máquinas virtuales	9	3	VM_SERVER_BOOTUP_ERROR
Arranque de máquina virtual	10	5	VM_BOOT_REQUEST
Datos de conexión a máquina virtual	11	5	VM_CONNECTION_DATA
Error de arranque de máquina virtual	12	4	VM_BOOT_FAILURE
Apagar infraestructura	13	1	HALT
Solicitud datos de conexión VNC	14	10	QUERY_ACTIVE_VM_VNC_DATA
Datos de conexión VNC	15	10	ACTIVE_VM_VNC_DATA
Comando ejecutado	16	5	COMMAND_EXECUTED
Error de apagado de servidor de máquinas virtuales	17	3	VM_SERVER_SHUTDOWN_ERROR
Error de baja de servidor de máquinas virtuales	18	3	VM_SERVER_UNREGISTRATION _ERROR
Apagado forzoso de máquina virtual	19	5	DOMAIN_DESTRUCTION
Error en apagado forzoso de máquina virtual	20	4	DOMAIN_DESTRUCTION_ERROR
Cambiar configuración de servidor de máquinas virtuales	21	4	VM_SERVER_CONFIGURATION _CHANGE
Error en cambio de configuración de servidor de máquinas virtuales	22	3	VM_SERVER_CONFIGURATION _CHANGE_ERROR
Solicitud estado del repositorio	23	10	QUERY_REPOSITORY_STATUS
Estado del repositorio	24	10	REPOSITORY_STATUS
Desplegar imagen	25	5	DEPLOY_IMAGE
Error al desplegar imagen	26	4	IMAGE_DEPLOYMENT_ERROR
Imagen desplegada	27	5	IMAGE_DEPLOYED
Borrar imagen	28	4	DELETE_IMAGE_FROM_SERVER
Error al borrar imagen	29	3	DELETE_IMAGE_FROM _SERVER_ERROR

CUADRO 4.8: Características principales de los paquetes utilizados en el servidor de *cluster* (parte 1)

Tipo de paquete	Código	Prioridad	Constante (tipo enumerado)
Crear imagen	30	4	CREATE_IMAGE
Error al crear imagen	31	3	IMAGE_CREATION_ERROR
Imagen creada	32	4	IMAGE_CREATED
Editar imagen	33	4	EDIT_IMAGE
Imagen editada	34	4	IMAGE_EDITED
Error al editar imagen	35	3	IMAGE_EDITION_ERROR
Borrar imagen de infraestructura	36	4	DELETE_IMAGE_FROM _INFRASTRUCTURE
Error al borrar imagen de infraestructura	37	3	DELETE_IMAGE_FROM _INFRASTRUCTURE_ERROR
Despliegue automático	38	4	AUTO_DEPLOY
Error de despliegue automático	39	3	AUTO_DEPLOY_ERROR
Solicitud de estadísticas de servidores de máquinas virtuales	40	10	QUERY_VM_SERVERS _RESOURCE_USAGE
Estadísticas de servidores de máquinas virtuales	41	10	VM_SERVERS_RESOURCE_USAGE
Reinicio forzoso de máquina virtual	42	5	DOMAIN_REBOOT
Error en reinicio forzoso	43	4	DOMAIN_REBOOT_ERROR

CUADRO 4.9: Características principales de los paquetes utilizados en el servidor de *cluster* (parte 2)

El resto de paquetes contienen un identificador de petición y un código de error.

Finalmente,

- los identificadores de petición, los identificadores de máquinas virtuales, los nombres y las direcciones IP son *strings*,
- el resto de datos de los paquetes son valores enteros.

4.5.7.16. Esquema de la base de datos

La base de datos del servidor de *cluster* contiene las siguientes tablas:

- `VMServer`. Esta tabla almacena los datos básicos de todos los servidores de máquinas virtuales registrados. Incluye las siguientes columnas:
  - `serverId`: identificador único del servidor de máquinas virtuales. Se trata de un valor entero, y es la clave primaria de la tabla.
  - `serverName`: nombre del servidor de máquinas virtuales. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud.
  - `serverStatus`: estado del servidor de máquinas virtuales. Ocupa un *byte*, y se codifica de acuerdo al cuadro 4.10.

    Para evitar errores, en el código fuente no manipulamos directamente estos valores: lo hacemos a través del tipo enumerado `SERVER_STATE_T`, definido en el paquete `clusterServer.database`.
  - `serverIP`: dirección IP del servidor de máquinas virtuales. Se trata de una cadena de caracteres de 15 *bytes* de longitud.
  - `serverPort`: puerto en el que escucha el servidor de máquinas virtuales.
  - `isEditionServer`: ocupa un *bit*, e indica si el servidor de máquinas virtuales es un servidor de edición o no.

Código	Descripción	Constante tipo enumeado (SERVER_STATE_T)
0	Arranque iniciado	BOOTING
1	Listo para atender peticiones	READY
2	Apagado	SHUT_DOWN
3	Reestableciendo conexión	RECONNECTING
4	*Timeout* al restablecer conexión	CONNECTION_TIMED_OUT

CUADRO 4.10: Codificación del estado de un servidor

- `ImageOnServer`. Esta tabla almacena la distribución de las imágenes. Contiene las siguientes columnas:
  - `serverId`: identificador único de un servidor de máquinas virtuales. Se trata de un valor entero.
  - `imageID`: identificador único de una imagen. Se trata de un valor entero que, junto con el identificador del servidor de máquinas virtuales, es la clave primaria de la tabla.
  - `status`: estado de la copia de la imagen. Ocupa un *byte*, y su valor se codifica de acuerdo al cuadro 4.11.

    Para evitar errores, en el código fuente no manipulamos directamente estos valores: lo hacemos a través del tipo enumerado `IMAGE_STATE_T`, definido en el paquete `clusterServer.database`.

Código	Descripción	Constante tipo enumeado (SERVER_STATE_T)
0	Copia lista para ser utilizada	READY
1	Copia con datos antiguos	EDITED
2	Copia en proceso de despliegue	DEPLOY
3	Copia pendiente de borrado	DELETE

CUADRO 4.11: Codificación del estado de una imagen

- `VMServerStatus`. Esta tabla almacena el estado de todos los servidores de máquinas virtuales. Contiene las siguientes columnas:
  - `serverId`: identificador único de un servidor de máquinas virtuales. Se trata de un valor entero, y es la clave primaria de la tabla.
  - `guests`: número de máquinas virtuales activas alojadas en el servidor. Se trata de un valor entero.
  - `ramInUse`: memoria RAM utilizada (en *kilobytes*). Se trata de un valor entero.
  - `ramSize`: tamaño de la memoria RAM instalada (en *kilobytes*). Se trata de un valor entero.
  - `freeStorageSpace`: espacio de almacenamiento libre (en *kilobytes*). Se trata de un valor entero.
  - `availableStorageSpace`: espacio de almacenamiento total (en *kilobytes*). Se trata de un valor entero.
  - `freeTemporarySpace`: espacio de uso temporal libre (en *kilobytes*). Se trata de un valor entero.
  - `availableTemporarySpace`: espacio de uso temporal total (en *kilobytes*). Se trata de un valor entero.
  - `activeVCPUs`: número de CPUs virtuales en uso. Ocupa un *byte*.

  - `physicalCPUs`: número de CPUs físicas del servidor de máquinas virtuales. Ocupa un *byte*.

- `AllocatedVMServerResources`. Esta tabla almacena los recursos reservados en los servidores de máquinas virtuales. Contiene las siguientes columnas:

  - `serverId`: identificador único de un servidor de máquinas virtuales. Se trata de un valor entero.
  - `commandID`: identificador único de la petición. Se trata de una cadena de caracteres de 70 *bytes* de longitud que es la clave primaria de la tabla.
  - `ramInUse`: memoria RAM reservada (en *kilobytes*). Se trata de un valor entero, y es la clave primaria de la tabla.
  - `freeStorageSpace`: espacio de almacenamiento reservado (en *kilobytes*). Se trata de un valor entero.
  - `freeTemporarySpace`: espacio de uso temporal reservado (en *kilobytes*). Se trata de un valor entero.
  - `activeVCPUs`: número de CPUs virtuales reservadas. Ocupa un *byte*.
  - `physicalCPUs`: número de CPUs físicas. Ocupa un *byte*.
  - `activeHosts`: número de máquinas virtuales. Ocupa un *byte*.
  - `remove`: este *bit* indica si la entrada puede eliminarse en la próxima actualización del estado.

- `VMFamily`. Esta tabla almacena las características de las familias de máquinas virtuales. Contiene las siguientes columnas:

  - `familyId`: identificador único de la familia. Se trata de un valor entero, y es la clave primaria de la tabla.
  - `familyName`: se trata de una cadena de caracteres de hasta 20 *bytes* de longitud. Esta columna contiene el nombre de la familia de máquinas virtuales.
  - `ramSize`: tamaño de la memoria RAM (en *kilobytes*). Se trata de un valor entero.
  - `osDiskSize`: tamaño de la imagen de disco del sistema operativo (en *kilobytes*). Se trata de un valor entero.
  - `dataDiskSize`: tamaño de la imagen de disco que almacenará los datos temporales y el fichero de paginación (en *kilobytes*). Se trata de un valor entero.
  - `vCPUs`: número de CPUs virtuales.Ocupa un *byte*.

- `VanillaImageFamilyOf`. Esta tabla asocia a cada imagen una familia de máquinas virtuales. Contiene dos columnas:

  - `familyId`: identificador único de la familia de máquinas virtuales. Se trata de un valor entero.
  - `imageID`: identificador único de la imagen de disco. Se trata de un valor entero.

  Las dos columnas de la tabla forman la clave primaria.

- `VanillaImageFamilyOfNewVM`. Esta tabla asocia a cada imagen de nueva creación una familia de máquinas virtuales. Contiene dos columnas:

  - `familyId`: identificador único de la familia de máquinas virtuales. Se trata de un valor entero.
  - `temporaryID`: se trata de una cadena de caracteres de hasta 70 *bytes* de longitud. Esta columna almacenará el identificador temporal de la nueva imagen, y será la clave primaria de la tabla.

- `ImageRepositoryStatus`. Esta tabla almacena el estado del repositorio de imágenes. Incluye las siguientes columnas:
  - `repositoryIP`: dirección IP del repositorio de imágenes. Se trata de una cadena de caracteres de 15 *bytes* de longitud.
  - `repositoryPort`: puerto de la conexión de control del repositorio de imágenes. Se trata de un valor entero.
  - `freeDiskSpace`: espacio de almacenamiento libre (en *kilobytes*). Se trata de un valor entero.
  - `availableDiskSpace`: espacio de almacenamiento total (en *kilobytes*). Se trata de un valor entero.
  - `status`: estado del repositorio de imágenes. Ocupa un *byte*, y puede tomar cualquiera de los valores del cuadro 4.10.
- `AllocatedImageRepositoryResources`. Esta tabla almacena los recursos reservados en el repositorio de imágenes. Sus columnas son las siguientes:
  - `commandID`: identificador único de la petición. Se trata de una cadena de caracteres de 70 *bytes* de longitud que es la clave primaria de la tabla.
  - `allocatedDiskSpace`: espacio de almacenamiento reservado (en *kilobytes*). Se trata de un valor entero.
  - `remove`: este *bit* indica si la entrada puede eliminarse en la próxima actualización del estado.
- `VMBootCommand`. Esta tabla almacena el instante de tiempo en el se enviaron los distintos comandos de arranque de máquinas virtuales. Contiene las siguientes columnas:
  - `commandID`: identificador único de la petición. Se trata de una cadena de caracteres de 70 *bytes* de longitud que es la clave primaria de la tabla.
  - `dispatchTime`: instante de tiempo en el que se envió el comando. Se trata de un valor en punto flotante.
  - `imageID`: identificador de la imagen que utiliza la máquina virtual a arrancar. Se trata de un valor entero.
- `ActiveVMDistribution`. Esta tabla almacena la distribución de todas las máquinas virtuales activas. Contiene las siguientes columnas:
  - `vmID`: identificador único de la máquina virtual. Se trata de una cadena de caracteres de 70 *bytes* de longitud que es la clave primaria de la tabla.
  - `serverID`: identificador único del anfitrión de la máquina virtual.
- `ImageEditionCommand`, `ImageDeletionCommand`, `AutoDeploymentCommand`. Estas tablas se utilizan para procesar los comandos de edición, borrado total y despliegue automático de imágenes, y contienen las siguientes columnas:
  - `commandID`: identificador único de la petición de edición, borrado o despliegue automático. Se trata de una cadena de caracteres de hasta 70 *bytes* de longitud, y es la clave primaria de la tabla.
  - `imageID`: identificador único de la imagen afectada por dicha petición.

Finalmente, las tablas `VMServer`, `ImageOnServer`, `VanillaImageFamily`, `VanillaImageFamilyOf` y `VanillaImageFamilyOfNewVM` residen en disco. El resto de tablas de la base de datos residen en memoria.

4.5.8. El paquete `clusterEndpoint`

Como ya hemos visto, el servidor de *cluster* puede llegar a mantener un elevado número de conexiones de red con los servidores de máquinas virtuales.

Cuanto menor sea el número de conexiones de red existentes entre el servidor web y el servidor de *cluster*, mayor podrá ser el número de servidores de máquinas virtuales a los que este puede atender y, por tanto, mayor podrá ser el número de usuarios que pueden utilizar el mismo *cluster* simultáneamente.

Por otra parte, la infraestructura de *CygnusCloud* se utilizará a través de una aplicación web. La implementación de la misma se simplifica considerablemente si se interactúa con la infraestructura a través de una base de datos.

Por tanto, al abordar el desarrollo de la aplicación web, decidimos

- reducir el número de conexiones de red entre el servidor web y el servidor de *cluster*, utilizando una única conexión para enviar todas las peticiones al servidor de *cluster*, y
- utilizar una base de datos para interactuar con la infraestructura de *CygnusCloud*.

Para que esto sea posible, el servidor web albergará un demonio, al que llamaremos *endpoint* de la aplicación web o, abreviadamente, proceso *endpoint* o *endpoint*. Este proceso se conectará a un servidor de *cluster*, extraerá las peticiones de los usuarios de una base de datos, las enviará al servidor de *cluster* y procesará las respuestas del servidor de *cluster*. En esta sección, mostraremos detalladamente su diseño.

4.5.8.1. La base de datos del *endpoint*

Para que los alumnos y los profesores puedan arrancar máquinas virtuales y modificar las imágenes disponibles en el *cluster*, la aplicación web necesita conocer, entre otras cosas,

- las características de todas las familias de máquinas virtuales,
- las características de todas las imágenes existentes en el *cluster*, es decir, su nombre, descripción, identificador y la familia de máquinas virtuales asociada, y
- la distribución de las imágenes de disco en el *cluster*.

Por otra parte, para que los administradores puedan hacer su trabajo, la aplicación web también necesita tener acceso a la configuración básica y al estado de todas las máquinas que forman parte del *cluster*.

Esta información se almacenará en una base de datos residente en el servidor *web*, a la que llamaremos base de datos del *endpoint*.

El *endpoint* deberá actualizar esta base de datos a medida que los usuarios realicen peticiones y la infraestructura las procese.

Es importante notar que la aplicación *web* nunca modificará directamente esta base de datos: siempre lo hará a través del *endpoint*.

4.5.8.2. La base de datos de comandos

La base de datos del *endpoint* permite a la aplicación web extraer toda la información necesaria para construir las peticiones que se enviarán a la infraestructura.

Ahora bien, si utilizamos exclusivamente esa base de datos, la aplicación web no podrá enviar dichas peticiones a la infraestructura ni obtener sus resultados. Para hacer esto posible, el servidor web alberga una base de datos especial, la base de datos de comandos. Ejecutando consultas y actualizaciones sobre ella, la aplicación web podrá enviar peticiones a la infraestructura y obtener sus resultados.

Por otra parte, dado que la aplicación web no se conecta directamente al servidor de *cluster*, el proceso *endpoint* también se encargará de

- extraer de la base de datos de comandos todas las peticiones emitidas por la aplicación web y enviarlas a la infraestructura, y de
- almacenar en la base de datos de comandos las respuestas que la infraestructura devuelve como resultado de todas esas peticiones.

4.5.8.3. Control de acceso

Para acotar la complejidad del proceso *endpoint*, a la hora de enviar las peticiones al servidor de *cluster* este no realiza ningún tipo de control de acceso. Por tanto,

- no realiza distinción alguna entre alumnos, profesores, administradores,...
- no distingue asignaturas, grupos de clase,...
- no limita el tipo de las operaciones que pueden enviar los usuarios a la infraestructura. Por ejemplo, el *endpoint* permite a un alumno apagar todos los servidores de la infraestructura.

Naturalmente, en el sistema *CygnusCloud* sí existen restricciones de acceso, y la aplicación web se ocupará de garantizar su cumplimiento.

Por otra parte, para hacer más robusto el sistema debemos evitar que los usuarios puedan enviar a la infraestructura un número arbitrario de peticiones por unidad de tiempo. Puesto que el proceso *endpoint* es el que envía estas peticiones a la infraestructura, lo más adecuado es hacer que este también limite el número de peticiones por unidad de tiempo que cada usuario puede enviar a la infraestructura.

4.5.8.4. Identificación de las peticiones

Al exponer el diseño del servidor de máquinas virtuales y del servidor de *cluster*, hemos mencionado en múltiples ocasiones el uso del identificador único de la petición.

Todas las peticiones que el *endpoint* envía a la infraestructura tienen asociado un identificador único, que se generará durante la creación de las mismas.

Estos identificadores son cadenas de caracteres de hasta 70 *bytes* de longitud, cuyo contenido es de la forma

```
UserID|Timestamp
```

donde `UserID` es el identificador del usuario que ha generado la petición, y `Timestamp` es el número de milisegundos transcurridos desde el 1 de enero de 1970 con una precisión de dos dígitos decimales.

Puesto que los servidores de máquinas virtuales y los servidores de *cluster* nunca descomponen ni modifican los identificadores únicos de las peticiones que reciben, es posible modificar la forma en que se generan estos identificadores de forma sencilla.

4.5.8.5. Envío de peticiones

El diagrama de clases de la figura 4.90 muestra las clases que intervienen en el envío de peticiones y sus relaciones.

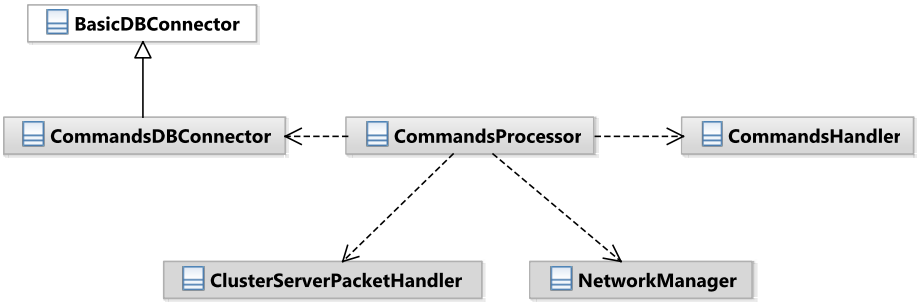

FIGURA 4.90: Clases que intervienen en el envío de peticiones a la infraestructura

En primer lugar, la clase `CommandsProcessor` dispone de métodos que extraen peticiones de la base de datos de comandos. Para ello, se utilizan los servicios de un objeto `CommandsDBConnector`. Como todas las clases que manipulan una base de datos, la clase `CommandsDBConnector` hereda de la clase `BasicDBConnector`, definida en el paquete `ccutils`.

Por otra parte, para enviar las peticiones a la infraestructura los objetos `CommandsProcessor` utilizan un objeto `NetworkManager` y un objeto `ClusterServerPacketHandler`. Como sabemos, la clase `ClusterServerPacketHandler` se define en el paquete `clusterServer.packetHandling`, y dispone de métodos para leer y crear los distintos tipos de paquete utilizados por los servidores de *cluster*.

Finalmente, para simplificar la implementación del *endpoint*, todas las peticiones que se almacenan en la base de datos de comandos están serializadas. Para deserializarlas, la clase `CommandsProcessor` utiliza los métodos de la clase `CommandsHandler`.

4.5.8.6. Procesamiento de las respuestas de la infraestructura

El diagrama de clases de la figura 4.91 muestra las principales clases que intervienen en el procesamiento de las respuestas de la infraestructura. Por claridad, de ahora en adelante abreviaremos `ClusterEndpoint` como `CE` en todos los diagramas UML.

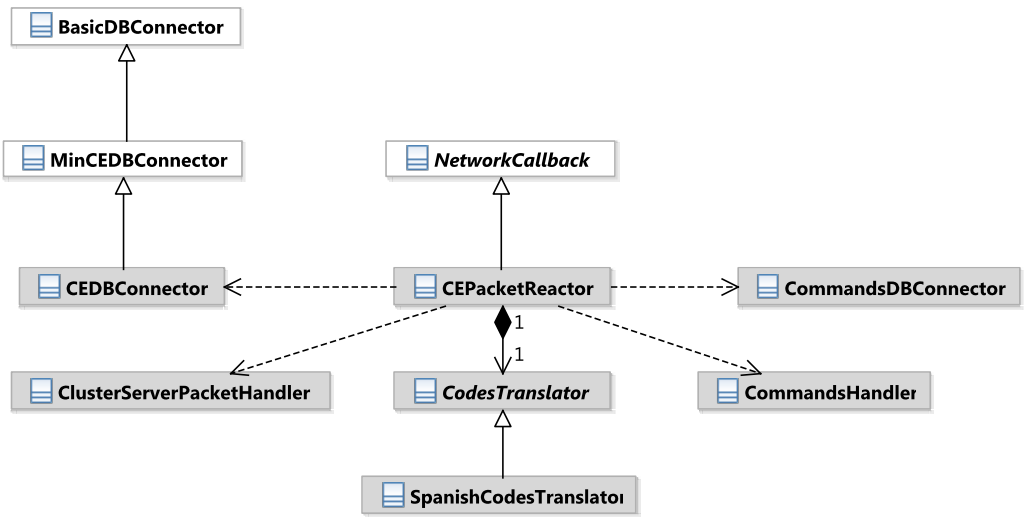

FIGURA 4.91: Clases que intervienen en el procesamiento de las respuestas de la infraestructura

Como podemos observar en el diagrama, la clase `ClusterEndpointPacketReactor` hereda de la clase abstracta `NetworkCallback`, por lo que procesará los paquetes enviados por el servidor de *cluster*. Para leerlos, utilizará un objeto `ClusterServerPacketHandler`.

Por otra parte, los códigos de error generados por la infraestructura y los códigos de estado de los servidores e imágenes no se muestran directamente a los usuarios: previamente, se convierten a *strings*. Para ello, el objeto ClusterEndpointPacketReactor utiliza un objeto SpanishCodesTranslator a través de la interfaz definida por la clase abstracta CodesTranslator.

Asimismo, cuando el servidor de *cluster* envía al servidor web un paquete con el resultado de una petición, estos datos deben almacenarse en la base de datos de comandos. Para ello, los objetos ClusterEndpointPacketReactor utilizan un objeto CommandsDBConnector y los métodos de serialización de resultados de la clase CommandsHandler.

Por otro lado, los objetos ClusterEndpointPacketReactor utilizarán un objeto ClusterEndpointDBConnector para actualizar la base de datos del *endpoint* tras recibir los paquetes correspondientes enviados por el servidor de *cluster*.

Finalmente, la clase ClusterEndpointDBConnector no hereda directamente de la clase BasicDBConnector, sino que lo hace a través de la clase MinCEDBConnector. Como veremos más adelante, la aplicación web utilizará los servicios de la clase MinCEDBConnector para hacer consultas sobre la base de datos del *endpoint*.

4.5.8.7. La clase ClusterEndpointEntryPoint

La clase ClusterEndpointEntryPoint es la clase principal del proceso *endpoint* del servidor web. Su principal responsabilidad es realizar los procesos de arranque y apagado del proceso *endpoint*. El diagrama de clases de la figura 4.92 muestra sus relaciones más importantes.

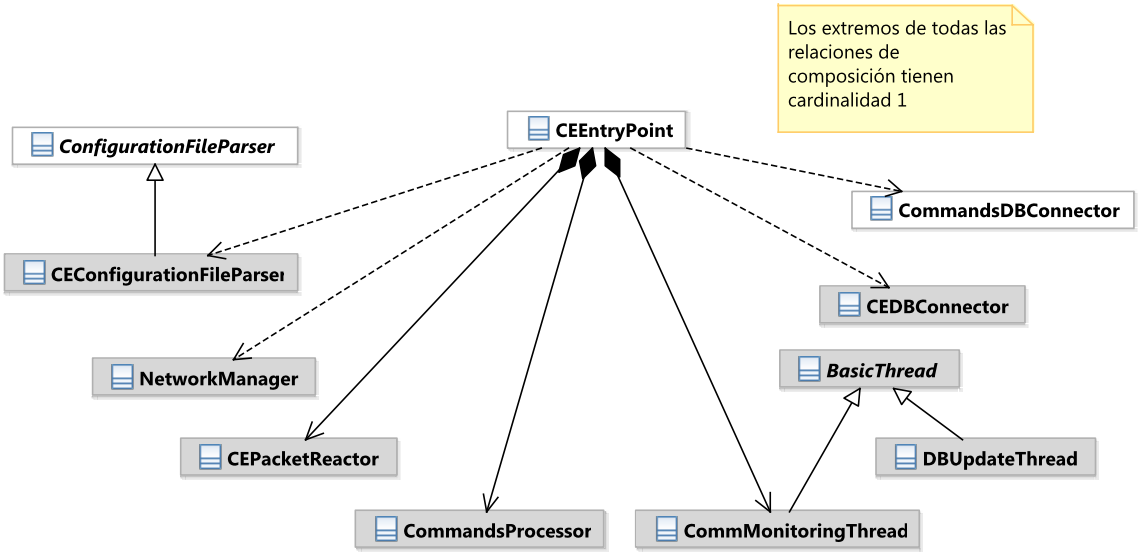

FIGURA 4.92: Principales relaciones de la clase ClusterEndpointEntryPoint

Para establecer la conexión con el servidor de *cluster*, los objetos ClusterEndpointEntryPoint utilizan un objeto NetworkManger.

Asimismo, para atender las peticiones de los usuarios y actualizar periódicamente la base de datos del *endpoint*, utilizan un objeto ClusterEndpointPacketReactor y un objeto CommandsProcessor.

Por otra parte, los objetos ClusterEndpointEntryPoint utilizan dos instancias de las clases de hilo DBUpdateThread y CommandsMonitoringThread. El primero permite iniciar la actualización periódica de la base de datos del *endpoint*, y el segundo permite generar errores por *timeout* durante el procesamiento de las peticiones.

Como la gran mayoría de las clases de hilo de la infraestructura, la clase DBUpdateThread y la clase CommandsMonitoringThread heredan de la clase BasicThread, definida en el paquete ccutils.

Asimismo, los objetos ClusterServerEndpoint utilizan un objeto ClusterEndpointConfigurationFileParser para procesar el fichero de configuración del proceso *endpoint*. Como sucedía con

el resto de *parsers* de ficheros de configuración, la clase `ClusterEndpointConfigurationFileParser` hereda de la clase abstracta `ConfigurationFileParser`, definida en el paquete `ccutils`.

Finalmente, el resto de dependencias del diagrama de clases de la figura 4.92 se deben al proceso de instanciación.

4.5.8.8. Arranque del proceso *endpoint*: secuencia básica

Para que la base de datos del *endpoint* y la base de datos de comandos se inicialicen correctamente, el proceso *endpoint* debe arrancarse antes que la aplicación web. El diagrama de secuencia de la figura 4.93 muestra los principales pasos de dicho proceso de arranque. Por claridad, estamos asumiendo que no se produce ningún error.

Como podemos observar en el diagrama de secuencia, se siguen los siguientes pasos:

1. se procesa el contenido del fichero de configuración.
2. se configuran las bases de datos y se establecen las conexiones con ellas.
3. se crea todo lo necesario para procesar los paquetes recibidos desde el servidor de *cluster* y para procesar las peticiones de los usuarios.
4. se establece la conexión con el servidor de *cluster*.
5. se inicializan los hilos de actualización de la base de datos del *endpoint* y de monitorización de peticiones.
6. el hilo principal pasa a procesar las peticiones enviadas por los usuarios a través de la aplicación web.

4.5.8.9. Arranque del proceso *endpoint*: tratamiento de errores

Durante el proceso de arranque del *endpoint*, se producirán errores si

- el contenido del fichero de configuración no es correcto.
- no es posible configurar las bases de datos.
- no es posible establecer la conexión con el servidor de máquinas virtuales.

Para tratar todos estos errores, se procede de la siguiente manera:

1. el hilo principal termina inmediatamente de procesar las peticiones de los usuarios.
2. si es necesario, se cierra la conexión de red con el servidor de *cluster*.
3. si es necesario, se detiene el hilo de actualización de la base de datos y el hilo de monitorización de peticiones.

Finalmente, es importante notar que el servicio de red se detendrá desde el hilo principal. Como vimos en la sección 4.5.5.7, si no lo detenemos así se producirá un cuelgue.

4.5.8.10. Actualización de la base de datos del *endpoint*

Para actualizar el contenido de la base de datos del *endpoint*, el proceso *endpoint* enviará periódicamente al servidor de *cluster* cinco paquetes de los tipos Solicitud de estado de servidores de máquinas virtuales, Solicitud de estadísticas servidores de máquinas virtuales, Solicitud de estado del repositorio, Solicitud de distribución de imágenes y Solicitud de datos de conexión VNC.

El diagrama de secuencia de la figura 4.94 muestra cómo se procesan estos paquetes. Por claridad, en él

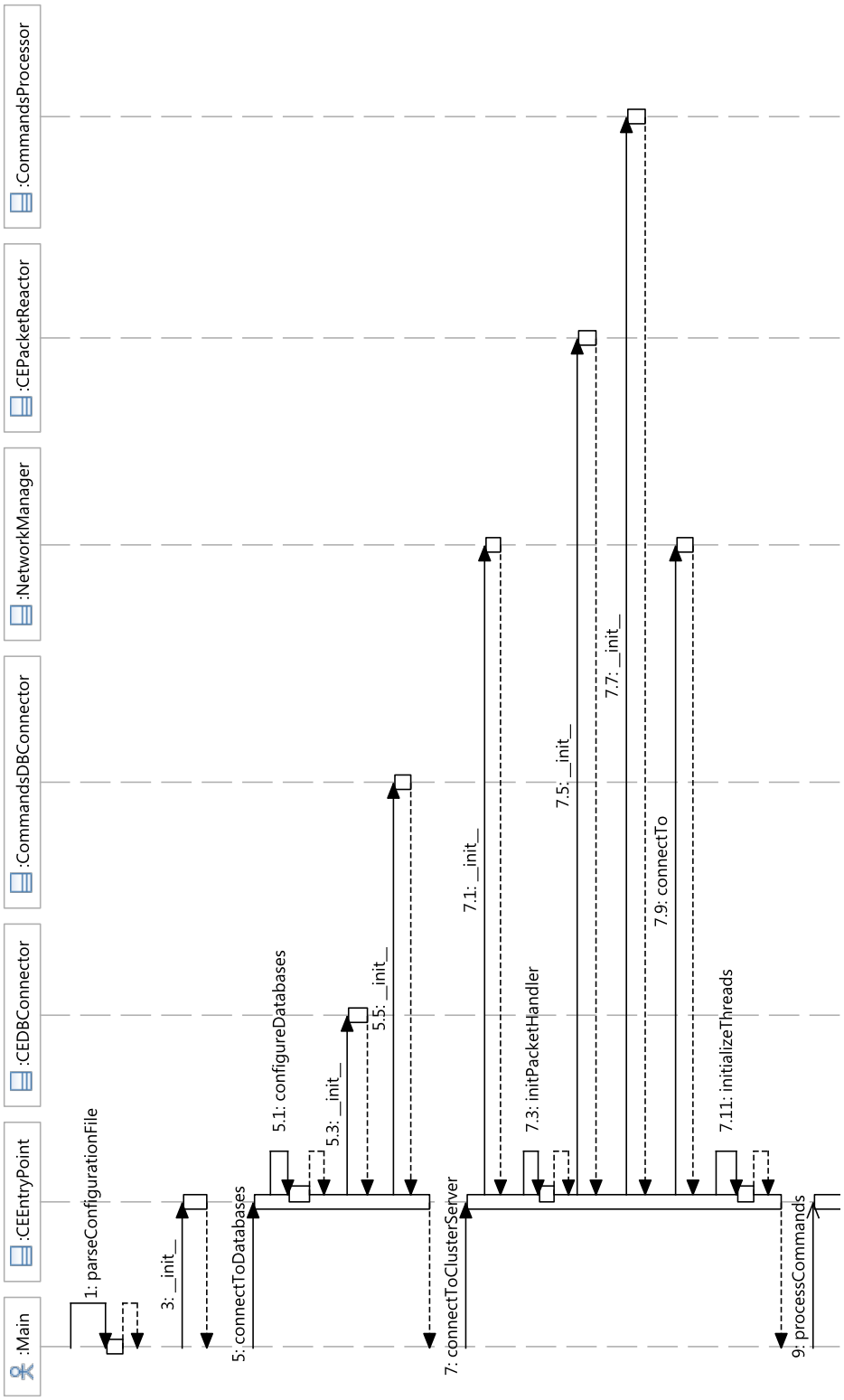

FIGURA 4.93: Arranque del proceso *endpoint*

- hemos abreviado la fase creación y de envío de los paquetes Solicitud de estado de servidores de máquinas virtuales, Solicitud de estadísticas servidores de máquinas virtuales, Solicitud de estado del repositorio, Solicitud de distribución de imágenes y Solicitud de datos de conexión VNC.
- sólo mostramos el procesamiento de dos respuestas: un paquete del tipo Estadísticas servidores de máquinas virtuales y un paquete del tipo Distribución de imágenes.

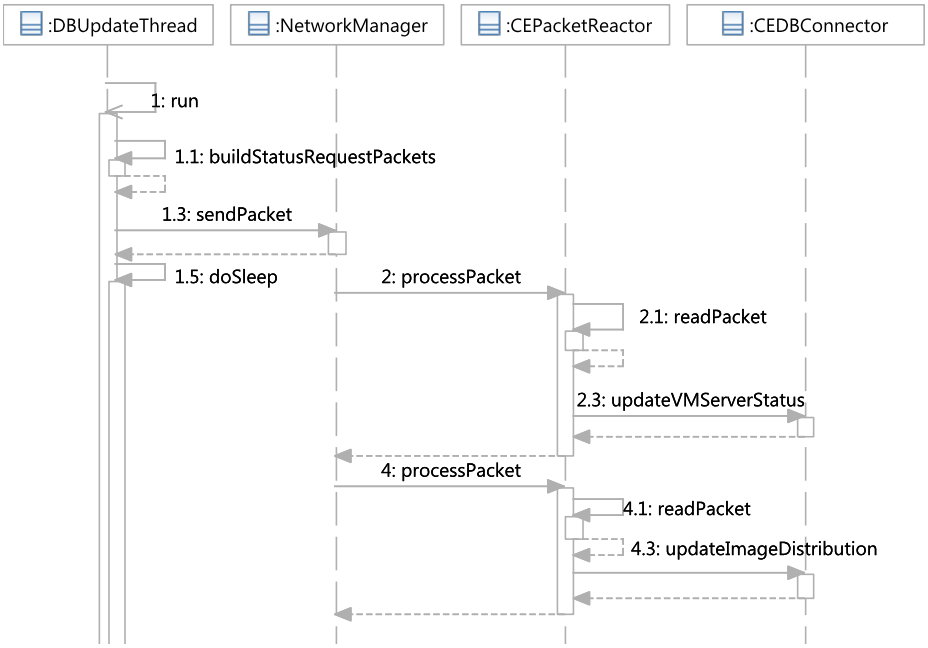

FIGURA 4.94: Actualización de la base de datos del *endpoint*

Como podemos observar en el diagrama de la figura 4.94,

1. tras crear y enviar los cinco paquetes, el hilo de actualización de la base de datos del *endpoint* dormirá hasta la siguiente actualización programada.

   El intervalo de actualización de la base de datos del *endpoint* se define en su fichero de configuración.

2. cuando se reciben los datos procedentes del servidor de *cluster*, estos se utilizan para actualizar el contenido de las tablas de la base de datos del *endpoint*.

4.5.8.11. Procesamiento de una petición: secuencia básica

Para procesar las peticiones de los usuarios,

1. el proceso *endpoint* las extrae de la base de datos de comandos y las deserializa.
2. acto seguido, el proceso *endpoint* inicia la interacción correspondiente con el servidor de *cluster*.
3. cuando recibe el paquete de respuesta procedente del servidor de *cluster*, el proceso *endpoint* genera la salida correspondiente, la serializa y la almacena en la base de datos de comandos.

El diagrama de secuencia de la figura muestra cómo se procesa una petición de arranque de una máquina virtual. Por claridad, a la hora de construir el diagrama hemos asumido que no se produce ningún error al procesar dicha petición.
Como podemos observar en el diagrama de secuencia, se siguen los siguientes pasos:

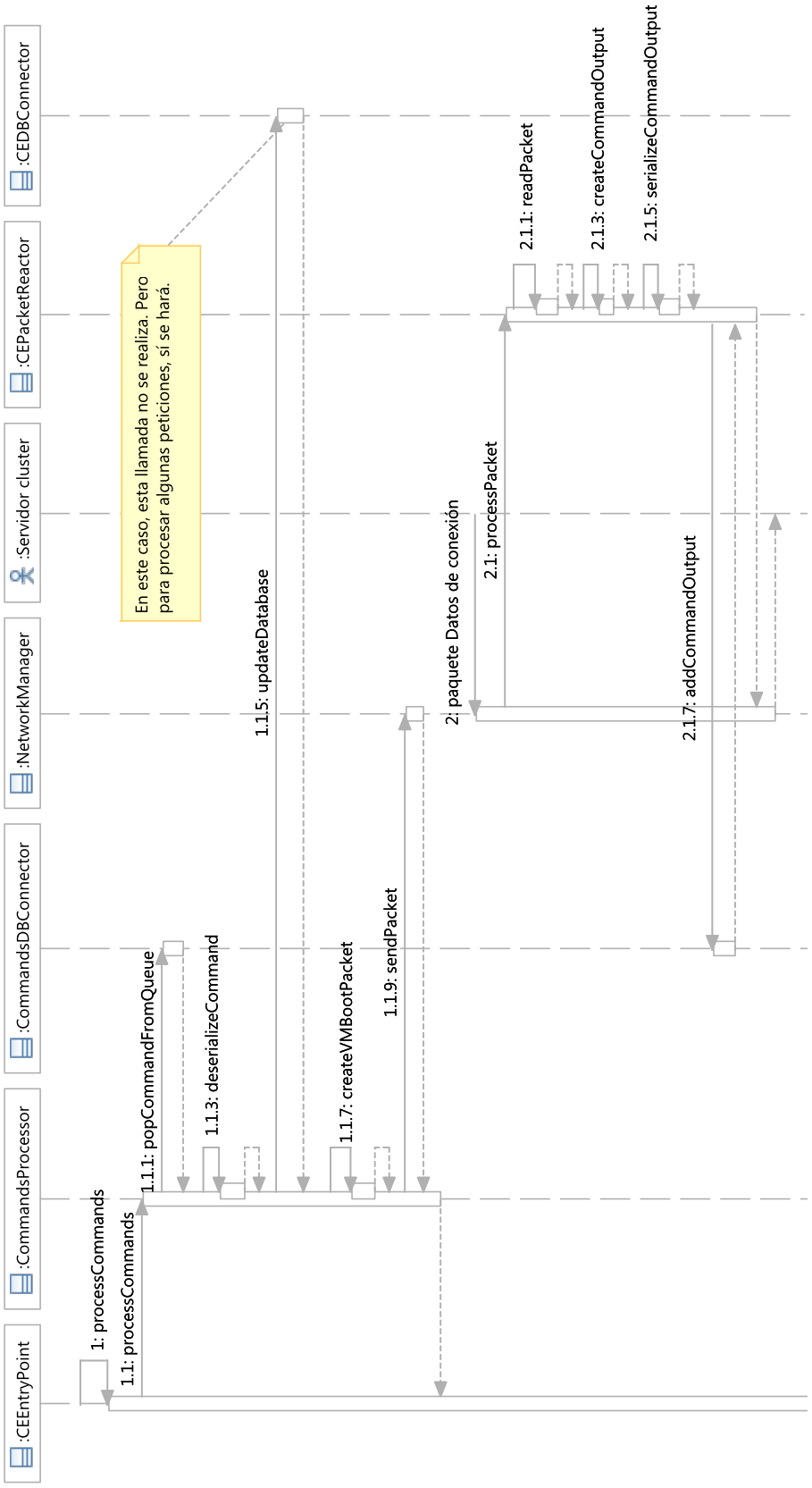

FIGURA 4.95: Arranque de una máquina virtual desde el *endpoint*

1. se extrae la petición de la base de datos de comandos. Todas las peticiones se encuentran almacenadas en una cola, y se atenderán por estricto orden de recepción.

2. se deserializa la petición para obtener sus datos.

3. se actualiza, si es necesario, la base de datos del *endpoint*. Para ello, se usan los datos asociados a la petición.

4. se crea un paquete del tipo Arranque de máquina virtual. Acto seguido, se envía dicho paquete al servidor de *cluster*.

5. el servidor de *cluster* responde a la petición enviando al *endpoint* un paquete del tipo Datos de conexión a máquina virtual.

6. utilizando la información contenida en ese paquete, se genera la salida de la petición.

   Es importante notar que todas las peticiones que envía un usuario tienen asociado un identificador único, y que este forma parte de las respuestas que devuelve la infraestructura. Esto permite

   - identificar las peticiones que han sido atendidas por la infraestructura, y
   - extraer los argumentos de la petición original de la base de comandos durante la generación de la salida.

7. finalmente, la salida de la petición se serializa y se añade a la base de datos de comandos. A partir de este momento, la aplicación *web* podrá leerla y utilizarla.

4.5.8.12. Procesamiento de una petición: tratamiento de errores

Cuando el procesamiento de la petición falla en el servidor de *cluster*, el *endpoint* recibirá un paquete de error.

En estos casos, la salida de la petición se genera de forma muy similar. No obstante, si el paquete recibido contiene un código de error, este se traducirá a *string* durante el proceso de generación de la salida.

4.5.8.13. Procesamiento de una petición: peticiones de alta latencia

Las peticiones de creación, edición, despliegue y borrado son peticiones de alta latencia: desde que se envían a la infraestructura hasta que terminan de ejecutarse pueden transcurrir horas.

Para facilitar la implementación de la aplicación web, las salidas de estas peticiones se almacenan de forma diferenciada en la base de datos de comandos. Así, de ahora en adelante, llamaremos **notificación** a la salida de una petición de alta latencia.

Las salidas de las peticiones de alta latencia se registran en la base de datos de comandos casi del modo que vimos en la sección 4.5.8.11. No obstante, a la hora de realizar la actualización de la base de datos de comandos, las salidas de estas peticiones se marcarán para distinguirlas del resto.

4.5.8.14. Apagado de toda la infraestructura

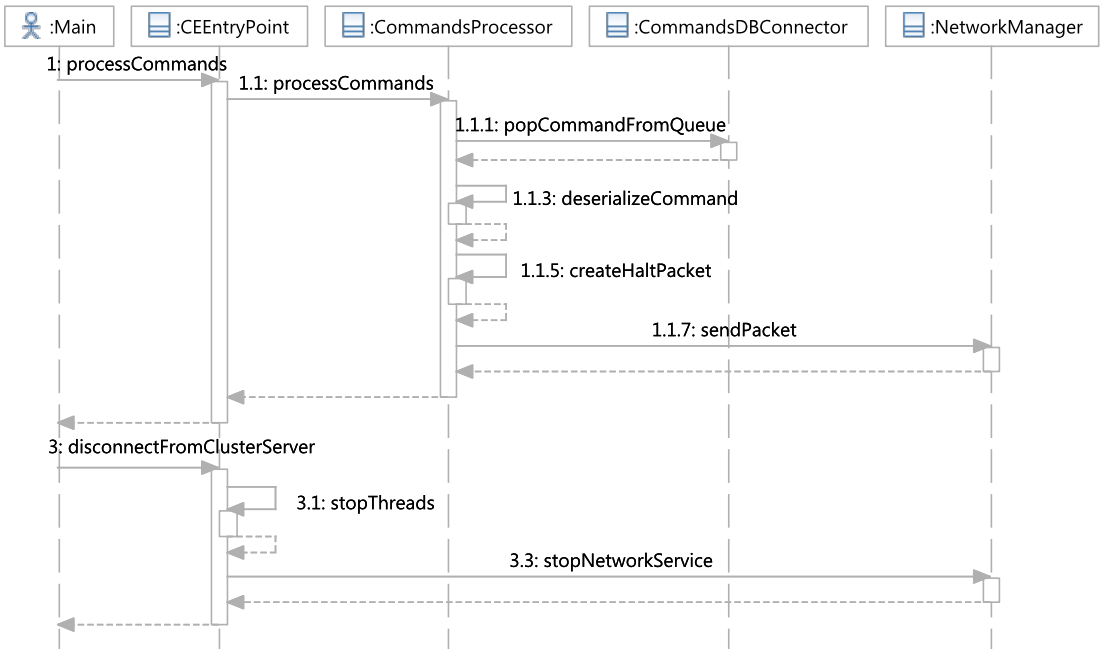

FIGURA 4.96: Apagado del proceso *endpoint*

Cuando desde la aplicación web se emite una petición de apagado de la infraestructura, se inicia el proceso de apagado de la infraestructura y del proceso *endpoint*. El diagrama de secuencia de la figura 4.96 muestra cómo se actúa en estos casos:

1. se crea un paquete del tipo Apagar infraestructura. Acto seguido, se envía ese paquete al servidor de *cluster*.
2. el hilo principal termina inmediatamente de procesar las peticiones de los usuarios.
3. acto seguido, el hilo principal
   a) detiene los hilos de actualización de la base de datos y de monitorización de peticiones, y
   b) cierra la conexión con el servidor de *cluster*.
4. el proceso *endpoint* termina. A medida que el servidor de *cluster* vaya enviando los paquetes correspondientes, las máquinas de la infraestructura también irán apagándose.

Es importante notar que el servicio de red debe detenerse desde el hilo principal. Si no lo hacemos así, se producirá un cuelgue. Justificamos esto en la sección 4.5.5.7.

4.5.8.15. Desconexión abrupta del servidor de *cluster*

El proceso *endpoint* es capaz de tratar una desconexión abrupta del servidor de *cluster*.

Para ello, la base de datos de comandos almacena el instante de tiempo en el que se inició el procesamiento de todas las peticiones generadas por los usuarios.

Como ya hemos mencionado, existe un hilo de monitorización de peticiones. Periódicamente, este hilo generará salidas de error para todas las peticiones que verifiquen

$$\text{instante actual} - \text{instante entrada} \geq \text{tiempo máximo}$$

donde el tiempo máximo se lee del fichero de configuración del proceso *endpoint*.

Naturalmente, cuando la salida de una petición se registra en la base de datos de comandos, el hilo de monitorización dejará de considerar dicha petición.

Por otra parte, tras detectar la desconexión del servidor de *cluster*, el *endpoint* iniciará el proceso de reconexión, utilizando para ello la funcionalidad de las clases del paquete `network`. Como dijimos en su momento, durante el proceso de reconexión se utilizará el algoritmo de retroceso exponencial binario truncado, con un máximo de 15 intentos. En la práctica, se intentará restablecer la conexión durante 15 minutos aproximadamente.

Finalmente, en caso de que no se pueda restablecer la conexión con el servidor de *cluster*, el proceso *endpoint* terminará.

4.5.8.16. Esquema de la base de datos de comandos

La base de datos de comandos contiene las siguientes tablas:

- `QueuedCommand`. En ella, se almacenan los identificadores de las peticiones que aún no se han atendido. Contiene las siguientes columnas:
  - `userID`: es el identificador único del usuario que ha creado la petición. Se trata de un valor entero.
  - `timestamp`. este valor de punto flotante contiene el instante de entrada de la petición representado como el número de milisegundos transcurridos desde el 1 de enero de 1970.
- `PendingCommand`. Esta tabla contiene los datos de todas las peticiones que se están ejecutando o que están a la espera de ser ejecutadas. Sus columnas son las siguientes:
  - `userID`, `timestamp`. Son idénticas a las de la tabla `QueuedCommand`.
  - `commandType`. Se trata de un *byte*, que codifica el tipo de la petición.
  - `commandArgs`. Se trata de una cadena de caracteres de hasta 100 *bytes* de longitud que contiene los argumentos de la petición serializados.
- `RunCommandOutput`. Esta tabla contiene las salidas asociadas a todas las peticiones que han terminado de ejecutarse. Sus columnas son las siguientes:
  - `userID`, `timestamp`. Son idénticas a las de la tabla `QueuedCommand`.
  - `outputType`. Se trata de un *byte*, que codifica el tipo de la salida.
  - `commandOutput`. Se trata de una cadena de caracteres de hasta 200 *bytes* de longitud que contiene la salida de la petición serializada.
  - `isNotification`. Se trata de un *bit* que indica si una fila de la tabla es la salida de un comando de alta latencia o no.

Todas estas tablas utilizan la clave primaria (`userID`, `timestamp`). Asimismo, por motivos de eficiencia, todas ellas residen en memoria.

4.5.8.17. Esquema de la base de datos del *endpoint*

Las principales tablas de la base de datos del *endpoint* son las siguientes:

- `VirtualMachineServer`. Esta tabla almacena los datos básicos de los servidores de máquinas virtuales pertenecientes al *cluster*. Contiene las siguientes columnas:
  - `serverName`. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud, que almacenará el nombre del servidor de máquinas virtuales.

  - `serverStatus`. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud, que almacenará la representación textual del estado del servidor de máquinas virtuales.
  - `serverIP`. Se trata de una cadena de caracteres de hasta 15 *bytes* de longitud, que almacenará la dirección IP del servidor de máquinas virtuales.
  - `serverPort`. Este valor entero almacenará el puerto de la conexión de control del servidor de máquinas virtuales.
  - `isEditionServer`. Se trata de un *bit*, que indica si el servidor de máquinas virtuales es un servidor de edición o no.

  La columna `serverName` es la clave primaria de esta tabla.

- `VirtualMachineServerStatus`. Esta tabla almacena el estado de todos los servidores de máquinas virtuales del *cluster*, y es idéntica a la tabla `VMServerStatus` de la base de datos del servidor de *cluster*.
- `ImageRepositoryStatus`. Esta tabla almacena el estado del repositorio de imágenes del *cluster*, y es idéntica a su tabla homónima de la base de datos del servidor de *cluster*.
- `ImageDistribution`. Esta tabla almacena la distribución de las imágenes. Contiene las siguientes columnas:
  - `serverName`. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud, que almacenará el nombre del servidor de máquinas virtuales.
  - `imageID`: identificador único de una imagen. Se trata de un valor entero que, junto con la columna `serverName`, es la clave primaria de la tabla.
  - `imageStatus`: Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud que almacenará la representación textual del estado de la copia de la imagen.
- `ActiveVirtualMachine`. Esta tabla contiene los datos asociados a todas las máquinas virtuales activas. Sus columnas son las siguientes:
  - `serverName`. Se trata de una cadena de caracteres de hasta 30 *bytes* de longitud, que almacenará el nombre del anfitrión de la máquina virtual.
  - `domainUID`. Se trata de una cadena de caracteres de hasta 70 *bytes* de longitud, que almacenará el identificador único de la máquina virtual. Esta columna es la clave primaria de la tabla.
  - `ownerID`. Se trata de un valor entero que almacena el identificador del propietario de la máquina virtual.
  - `imageID`. Es el identificador único de la imagen utilizada por la máquina virtual. Se trata de un valor entero.
  - `port`. Es el puerto del *websocket* asociado al servidor VNC de la máquina virtual. Se trata de un valor entero.
  - `password`. Se trata de una cadena de caracteres de 65 *bytes* de longitud que almacenará la contraseña del servidor VNC asociado a la máquina virtual.
- `VMFamily`. Esta tabla es idéntica a su tabla homónima de la base de datos del servidor de *cluster*.
- `Image`. Esta tabla almacena los datos básicos de una imagen de disco que no se está creando o editando. Sus columnas son las siguientes:
  - `imageID`. Es el identificador único de la imagen. Se trata de un valor entero, que es la clave primaria de la tabla.
  - `vmFamilyID`. Es el identificador único de la familia de máquinas virtuales asociada a la imagen. Se trata de un valor entero.

  - `name`. Se trata de una cadena de caracteres de hasta 20 *bytes* de longitud que almacenará el nombre de la imagen.
  - `description`. Se trata de una cadena de caracteres de hasta 200 *bytes* de longitud que almacenará la descripción de la imagen.
  - `osFamily`, `osVariant`. Estos valores ocupan un *byte*, y asignan una familia de sistemas operativos y un sistema operativo concreto a la imagen.
  - `isBaseImage`. Indica si la imagen es una imagen base o no.
  - `isBootable`. Indica si la imagen se puede utilizar para arrancar máquinas virtuales convencionales o no.

- `EditedImage`. Esta tabla almacena los datos básicos de una imagen de disco que se está creando o editando. Sus columnas son las siguientes:

  - `temporaryID`. Se trata de una cadena de caracteres de hasta 70 *bytes* de longitud, que almacenará el identificador único de la petición de creación o edición de imagen. Esta columna es la clave primaria de la tabla.
  - `imageID`. Es el identificador único asignado a la imagen. Se trata de un valor entero.
  - `vmFamilyID`. Es el identificador único de la familia de máquinas virtuales asociada a la imagen. Se trata de un valor entero.
  - `name`. Se trata de una cadena de caracteres de hasta 20 *bytes* de longitud, que almacenará el nombre de la imagen.
  - `description`. Se trata de una cadena de caracteres de hasta 200 *bytes* de longitud, que almacenará la descripción de la imagen.
  - `osFamily`, `osVariant`. Estos valores ocupan un *byte*, y asignan una familia de sistemas operativos y un sistema operativo concreto a la imagen.
  - `isBaseImage`. Indica si la imagen es una imagen base o no.
  - `isBootable`. Indica si la imagen se puede utilizar para arrancar máquinas virtuales convencionales o no.

Finalmente, sólo las tablas `Image`, `EditedImage`, `VMFamily`, `OSFamily` y `OSVariant` residen en disco. Todas las demás residen en memoria, y se rellenarán en cada arranque del proceso del *endpoint*.

4.5.9. El paquete `clusterConnector`

Al atender las peticiones de los usuarios, la aplicación web

- realiza consultas sobre la base de datos del *endpoint*, e
- inserta peticiones y extrae sus resultados en la base de datos de comandos.

Para ello, utiliza la clase `ClusterConnector`. Esta clase es lo último que hace falta para que la aplicación web pueda comunicarse con la infraestructura a través del *endpoint*.

De ahora en adelante, llamaremos **conector** al objeto `ClusterConnector` que utiliza la aplicación web.

4.5.9.1. Relaciones de la clase `ClusterConnector`

El diagrama de clases de la figura 4.97 muestra las relaciones de la clase `ClusterConnector`.

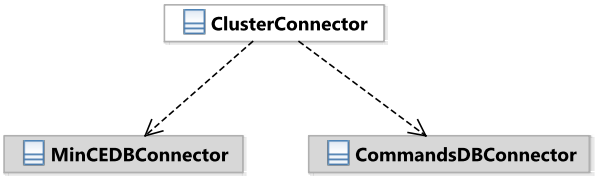

FIGURA 4.97: Relaciones de la clase `ClusterConnector`

Como podemos observar en él, los objetos `ClusterConnector` utilizan

- un objeto `CommandsDBConnector` para insertar peticiones y extraer los resultados de las mismas de la base de datos de comandos, y
- un objeto `MinimalClusterEndpointDBConnector` para realizar consultas sobre la base de datos del *endpoint*.

4.5.9.2. Envío de una petición a la infraestructura

En esta sección mostraremos cómo se insertan peticiones en la base de datos de comandos. Para ello, utilizaremos un ejemplo.

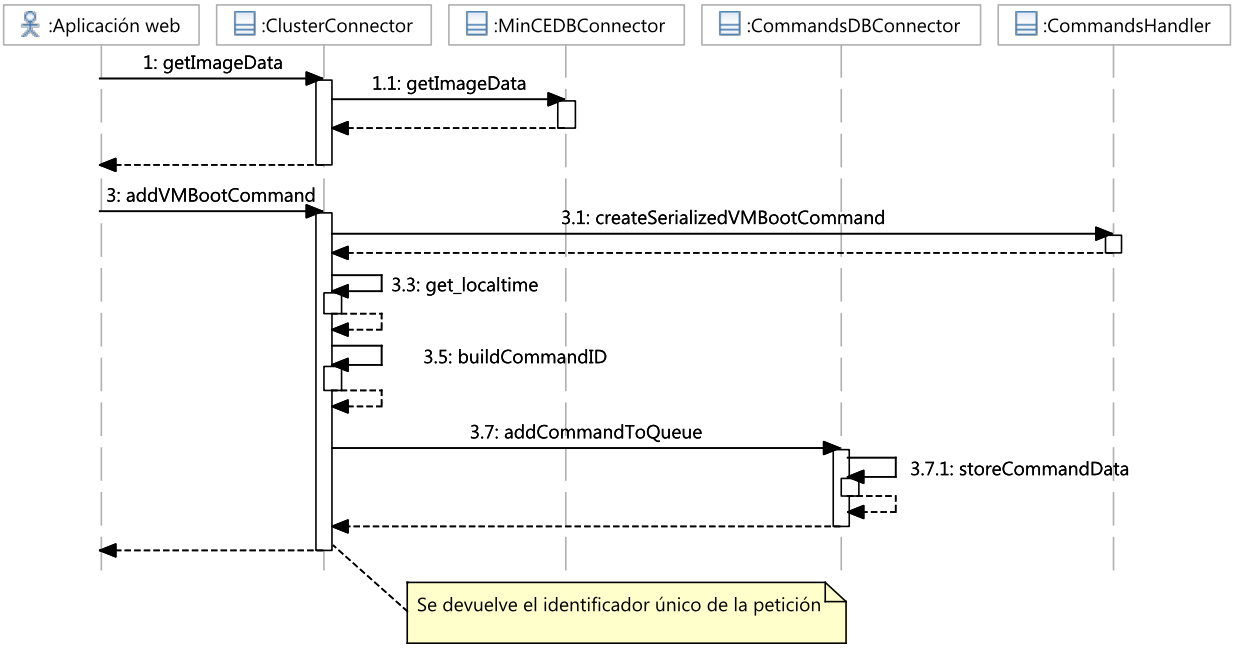

FIGURA 4.98: Envío de una petición a la infraestructura

El diagrama de secuencia de la figura 4.98 muestra el proceso de registro de una petición de arranque de una máquina virtual. En dicho diagrama, podemos distinguir los siguientes pasos:

1. la aplicación web realiza, a través del objeto `ClusterConnector`, las consultas necesarias para obtener la información que debe incluirse en la petición.

   Estas consultas se ejecutan sobre la base de datos del *endpoint*. En el caso del ejemplo, se consultan los datos de las imágenes para mostrárselos al usuario.

2. cuando el usuario escoge una imagen y decide arrancarla, la aplicación web solicita, a través del objeto `ClusterConnector`, la creación de una petición de arranque de una máquina virtual.

En la llamada al método correspondiente del objeto ClusterConnector, la aplicación web incluirá todo lo que necesita la infraestructura para atender la petición (en este caso, todo lo que hace falta para construir el paquete del tipo Arranque de máquina virtual que se enviará al servidor de *cluster*).

3. el objeto `ClusterConnector`,

   a) genera el identificador único de la nueva petición. Para ello, la aplicación web también le suministra el identificador del usuario que ha enviado la petición.

   b) serializa la petición de arranque de la máquina virtual.

   c) la inserta en la cola de peticiones, que forma parte de la base de datos de comandos.

   d) devuelve el identificador único de la petición a la aplicación web.

4.5.9.3. Obtención del resultado de la petición

El diagrama de secuencia de la figura 4.99 muestra cómo se extraen los resultados de dos peticiones de la base de datos de comandos. La primera es la salida de una petición normal, y la segunda es una notificación.

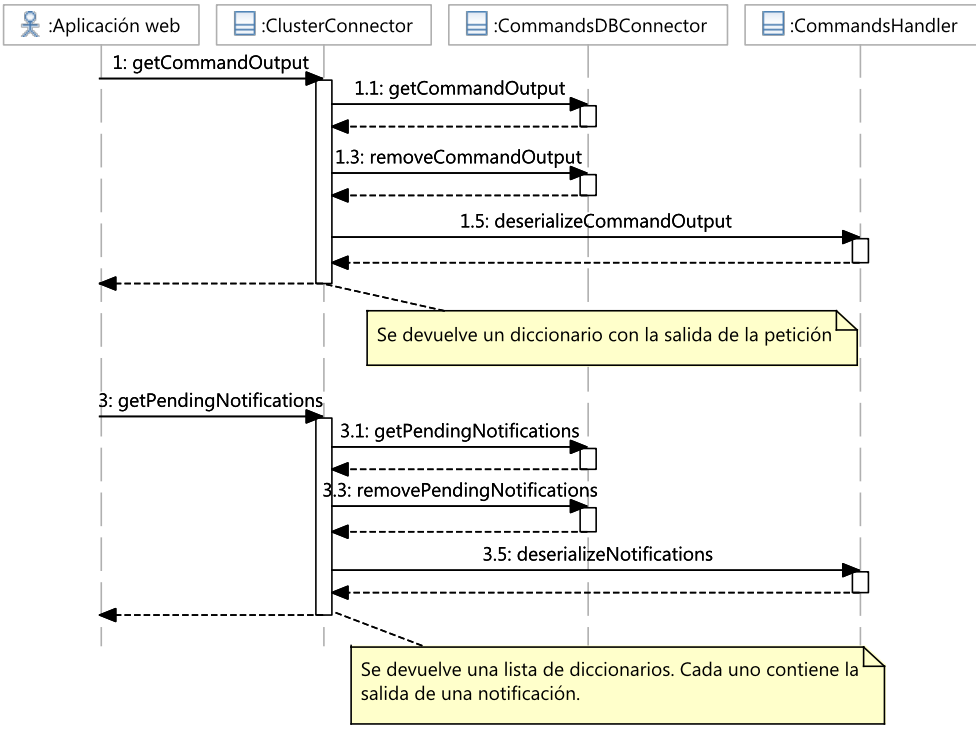

FIGURA 4.99: Obtención del resultado de la petición

Para obtener la salida asociada a una petición normal, la aplicación web utiliza el identificador único que obtuvo al enviar dicha petición a la infraestructura. Como podemos observar en el diagrama, la salida de la petición se genera de la siguiente manera:

1. se extrae y elimina la salida serializada de la base de datos.
2. se deserializa la salida de la petición.
3. se devuelve un diccionario con el contenido de la misma.

Por otra parte, la salidas de las peticiones de alta latencia o notificaciones se obtienen de forma similar. No obstante, de cara a facilitar la implementación de la aplicación web, las notificaciones no se extraen una a una: para obtenerlas, se utiliza un identificador de usuario.

4.5.10. El paquete webServer

En este paquete se trata todo lo relacionado con la gestión de la aplicación web de *CygnusCloud*. Así, en él, se incluyen todos los módulos encargados de gestionar las funcionalidades de interacción con los usuarios a través de la web y los ficheros `.html` y `.css` necesarios para la implementación de las páginas.

4.5.10.1. Modelo Vista-Controlador

Para llevar a cabo el desarrollo de la web ha sido necesario la utilización de un *framework* web que nos facilitase algunos aspectos y estructurase el sistema de forma correcta. Para ello, hemos optado por la utilización de `web2py` cuyas principales características se mencionaron en la sección 4.4.22.

`web2py` fuerza al desarrollador a utilizar unas buenas prácticas de ingeniería del software, evitando la repetición de código en la medida de lo posible y estandarizando la manera de hacer las cosas.

Por ello, este *framework* utiliza un modelo *vista-controlador* que incentiva al desarrollador a separar la forma de representar los datos(la vista) y el flujo de trabajo de la aplicación(el controlador). Este modelo ofrece una estructura modular que permite aislar los posibles errores en una sección de código concreta y trabajar en paralelo de forma independiente, reduciendo así los posibles tiempos de espera en el desarrollo de la web.

El flujo de trabajo típico de una petición `web2py` viene representado en el diagrama 4.100.

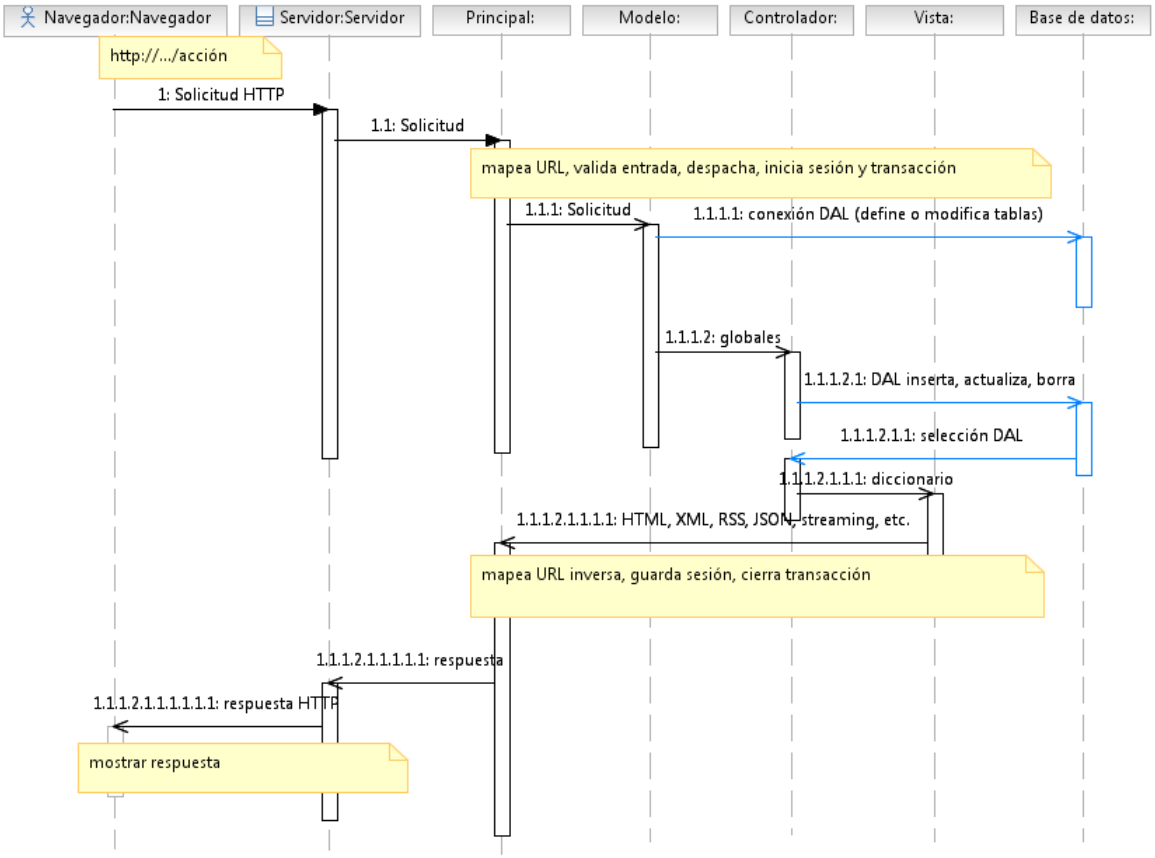

FIGURA 4.100: Procesamiento de una petición en `web2py`

En este diagrama podemos destacar los siguientes elementos:

- El navegador, a partir del cual el usuario se conectará a la web.

- El servidor, puede ser el servidor incluido en web2py o un servidor de terceros, tal como Apache.
- La aplicación principal, encargada de realizar todas las tareas comunes tal y como la gestión de *cookies*, sesiones, transacciones y enrutamiento.
- Los modelos, vistas y controladores que componen la aplicación del desarrollador.
- La base de datos que se encargará de almacenar la información utilizada por la web.

Todas las llamadas se encierran en una transacción. Cualquier excepción no contemplada hace que la transacción se cancele.

Con web2py es posible registrar tareas recurrentes para ejecutarse en horas específicas y/o después de determinadas acciones. También es posible alojar varias aplicaciones en una misma instancia. [15]

4.5.10.2. Estructura general de la web

Como web2py estructura las aplicaciones siguiendo un modelo *vista-controlador*, en *CygnusCloud* ha sido necesario estructurar el código de la aplicación web en un grupo de control y un grupo de vista. A parte de ambos grupos también se han creado más grupos con ficheros estáticos, ficheros .css y .html, módulos importados y en general todos los ficheros necesarios para dar la funcionalidad y el aspecto requeridos en la web.

Con todo esto, la web de *CygnusCloud* puede subdividirse en los siguientes grupos según su finalidad y tipo:

- Vistas: Este grupo incluye todos los ficheros en formato .html encargados de establecer el aspecto de los componentes, su posicionamiento en las páginas y su comportamiento. Entendemos por comportamiento de un componente, su capacidad para realizar ciertas acciones que mejoren la interacción con los usuarios. Según las restricciones de web2py será necesario disponer de un fichero de vista por cada sección disponible en la web. Cada uno de estos ficheros de vista reciben los componentes creados en el controlador asociado, ajustan su aspecto al que se mostrará en la página, y los posicionan en el lugar deseado.
- Controladores: Este grupo incluye el conjunto de módulos python que aportan todas las funcionalidades que serán gestionadas por la web. *CygnusCloud* dispone de un módulo para cada uno de los tipos de usuarios que pueden acceder a la web (alumnos, administradores y profesores), un módulo para las páginas de acceso público, un módulo para la página que contiene el cliente VNC y un módulo appAdmin creado automáticamente por web2py para la gestión de las bases de datos. Dentro de cada módulo encontramos una función asociada a cada una de las secciones que forman la aplicación web. Hablaremos más en profundidad sobre este tema en la sección 4.5.10.4.
- Modelos: Este grupo incluye dos módulos python encargados de la creación de las bases de datos y las características generales de la aplicación. Así, el módulo db.py crea las tablas de la base de datos encargada de la gestión de la información en la web. El módulo menu.py define las características generales de la aplicación web, tales como su logotipo, su autor, sus *tags* de búsqueda... Este par de módulos son creados y exigidos por web2py.
- Lenguajes: Este grupo incluye los diccionarios de traducción de las páginas de la web. Estos diccionarios deben ser rellenados por el desarrollador y son usados internamente por web2py para traducir todas las palabras que aparezcan en los mismos.
- Archivos estáticos: Este grupo incluye los ficheros .css y .js encargados de definir la apariencia de la web, las imágenes y en general cualquier fichero que defina algún componente externo que se importará en la web.
- Módulos: En este apartado se incluyen todos los módulos python secundarios utilizados por los controladores. Así, en este grupo se encuentra el conector que permite la interacción entre el servidor web y el resto de la infraestructura.

4.5.10.3. Estructura de direcciones

Con el fin de poder gestionar la web de la forma más modular posible y aprovechando el flujo de direcciones usado por `web2py`, vamos a estructurar los diferentes apartados de la web en forma de árbol de direcciones, de forma que cada uno de sus nodos pueda realizar saltos a cualquier otro nodo del árbol según las acciones llevadas a cabo por los usuarios. Así, el elemento principal será la aplicación en sí (`/CygnusCloud`). A partir de esta dirección comenzarán a surgir las direcciones asociadas a las páginas manteniendo la estructura de secciones y subsecciones correspondientemente. De esta forma, las diferentes *URLs* sobre las que trabajará la aplicación web son:

- `/CygnusCloud/main/login`: Es la página de inicio de sesión donde el usuario introduce su nombre y contraseña para iniciar la sesión.
- `/CygnusCloud/main/about`: Es la página en la que se habla acerca de *CygnusCloud* y su finalidad.
- `/CygnusCloud/student/runVM`: Es la página de arranque de máquinas virtuales para los estudiantes.
- `/CygnusCloud/student/runningVM/stopVM`: Página encargada de detener máquinas virtuales en ejecución.
- `/CygnusCloud/student/runningVM/openVM`: Esta página permite abrir una máquina virtual en ejecución arrancada por el estudiante.
- `/CygnusCloud/student/showNotifications`: Página que muestra las notificaciones pendientes para los estudiantes.
- `/CygnusCloud/administrator/runVM/run`: Es la página de arranque de máquinas virtuales para los administradores.
- `/CygnusCloud/administrator/runVM/stop`: Es la página de detención de máquinas virtuales en ejecución para los administradores.
- `/CygnusCloud/administrator/runVM/open`: Página que permite a un administrador abrir una máquina en ejecución.
- `/CygnusCloud/administrator/runVM/edit`: Página que permite editar una imagen con respecto al servidor de máquinas virtuales donde se encuentra desplegada.
- `/CygnusCloud/administrator/servers/add_servers`: Es la página encargada de añadir nuevos servidores.
- `/CygnusCloud/administrator/servers/remove_servers`: Es la página encargada de eliminar servidores de máquinas virtuales existentes.
- `/CygnusCloud/administrator/servers/servers_state`: Página que muestra el estado en que se encuentra el repositorio y los servidores de máquinas virtuales activos.
- `/CygnusCloud/administrator/servers/stop_system`: Es la página que permite detener todos los servidores de la infraestructura.
- `/CygnusCloud/administrator/users/remove`: Esta es la página encargada de eliminar a usuarios previamente creados.
- `/CygnusCloud/administrator/users/add`: Es la página encargada de crear nuevos usuarios con las especificaciones determinadas por el administrador.
- `/CygnusCloud/administrator/users/associate_subjects`: Es la página encargada de establecer las relaciones entre un usuario previamente creado y un grupo de asignatura.
- `/CygnusCloud/administrator/subjects/add`: Es la página encargada de crear nuevos grupos de asignatura con la información introducida por el administrador.

- `/CygnusCloud/administrator/subjects/remove`: Es la página encargada de eliminar algún grupo de asignatura existente.
- `/CygnusCloud/administrator/showNotifications`: Página que muestra las notificaciones pendientes para este administrador.
- `/CygnusCloud/vncClient/VNCPage`: Página que contiene el escritorio de trabajo ejecutado por noVNC.
- `/CygnusCloud/teacher/runVM/run`: Esta página muestra las máquinas que pueden ser arrancadas por el profesor.
- `/CygnusCloud/teacher/runningVM/stopVM`: Página que permite detener una máquina virtual en ejecución para los profesores.
- `/CygnusCloud/teacher/runningVM/openVM`: Es la página de apertura de máquinas en ejecución para los profesores.
- `/CygnusCloud/teacher/createAndEdit/createVanillaVM`: Esta página permite a un profesor crear una nueva imagen a partir de otra existente.
- `/CygnusCloud/teacher/createAndEdit/editVM`: Esta página permite editar imágenes por parte de un profesor.
- `/CygnusCloud/teacher/associateSubjects`: Página que permite asociar una máquina virtual con un determinado grupo de asignatura.
- `/CygnusCloud/teacher/showNotifications`: Es la página que muestra las notificaciones pendientes para los profesores.

La figura 4.101 muestra la estructura de direcciones para los alumnos. De igual forma, la figura 4.102 muestra la estructura de direcciones para los profesores y la figura 4.103 para los administradores.

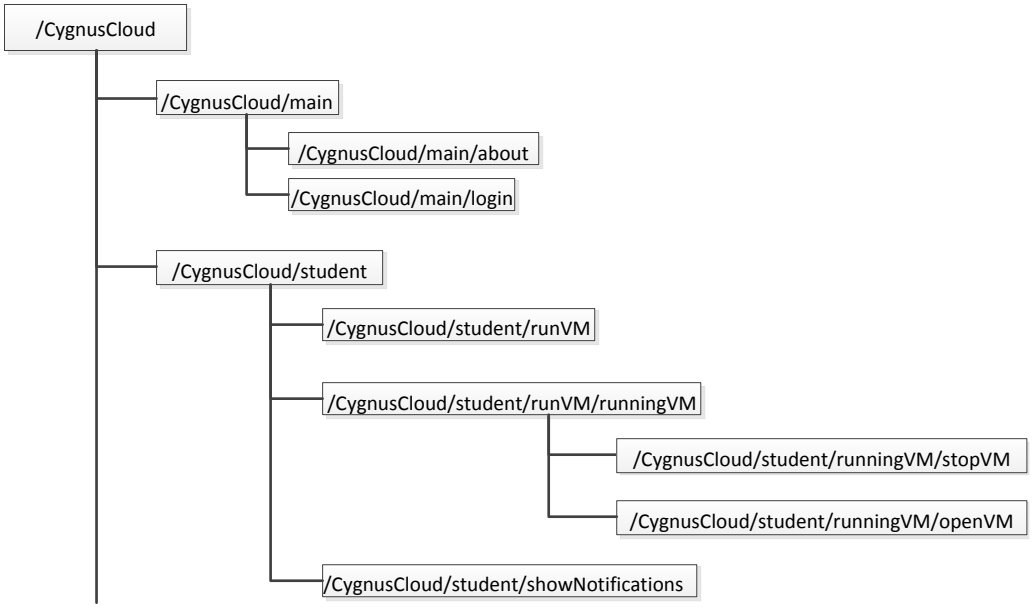

FIGURA 4.101: Estructura de direcciones de la web para los alumnos

FIGURA 4.102: Estructura de direcciones de la web para los profesores

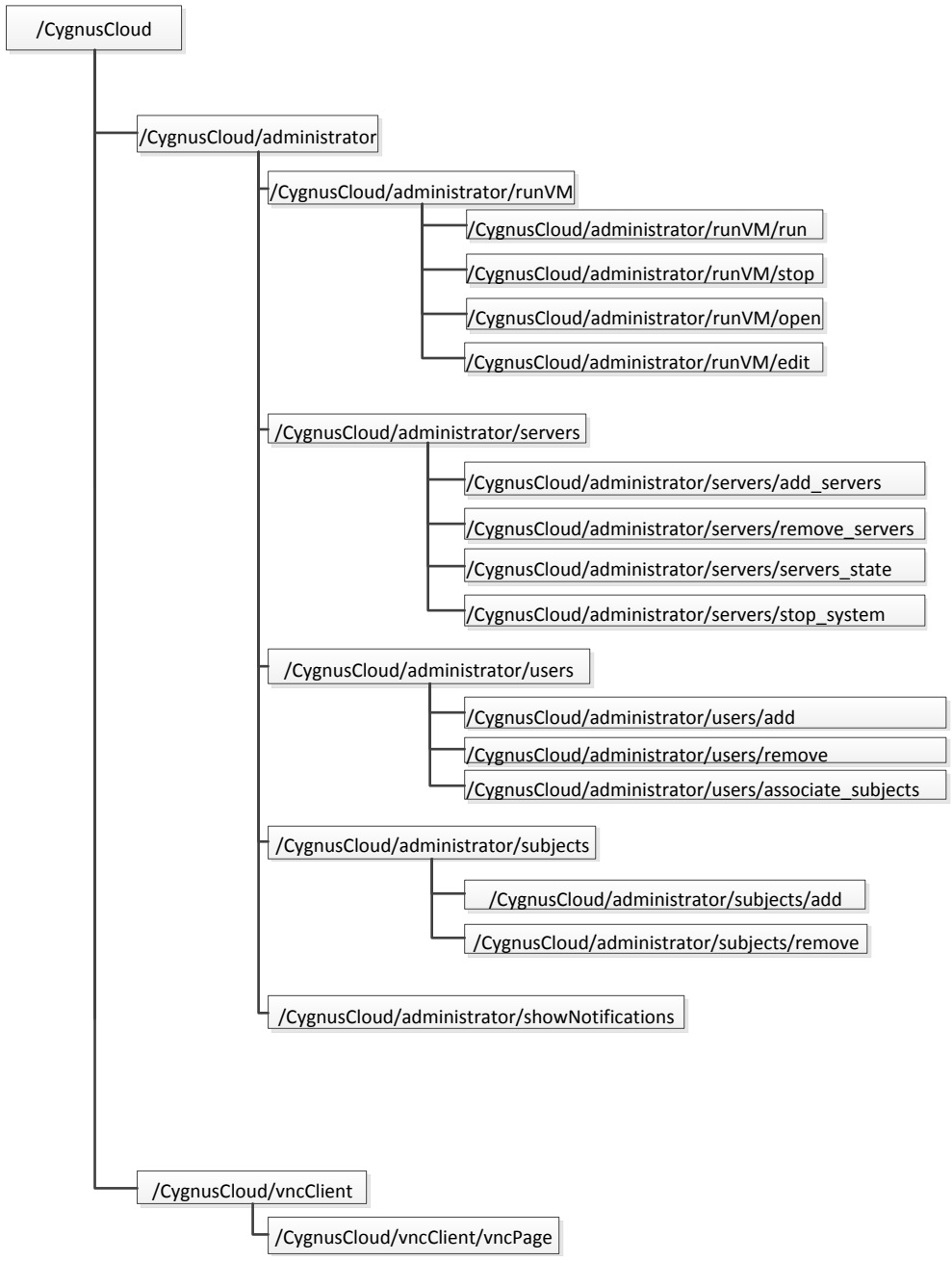

FIGURA 4.103: Estructura de direcciones de la web para los administradores

4.5.10.4. Secciones y subsecciones

Para poder hacer frente al elevado número de páginas que ofrece *CygnusCloud*, hemos optado por estructurarlas en un sistema de tres niveles, que permite filtrarlas teniendo en cuenta los privilegios de acceso y la finalidad de cada página.

El primer nivel se centra en los privilegios de acceso. *CygnusCloud* agrupa a los usuarios registrados en tres tipos según los privilegios de los que dispongan. Estos privilegios dan acceso a unas u otras páginas. Podemos encontrar los siguientes tipos de usuarios:

- Alumnos: En este tipo se incluyen todos los alumnos que cursan alguna de las asignaturas registradas en la web. Este tipo de usuarios es el más restringido, dándole únicamente acceso a la página de arranque de máquinas virtuales, asociadas a las asignaturas matriculadas por el alumno, y a la página de visualización, detención y apertura de sus máquinas arrancadas.

- Profesores: Este tipo incluye a todos los profesores que imparten alguna de las asignaturas registradas en la web. Dispone de unos privilegios de acceso mayores que el caso del alumno, pudiendo acceder, además de las páginas de arranque de máquinas virtuales y gestión de máquinas arrancadas, a páginas de creación, edición y borrado de imágenes.

- Administradores: Este tipo incluye a todos los técnicos y administradores encargados de la gestión total de la aplicación web. Es el tipo más privilegiado, teniendo acceso a las páginas de :

  - Arranque de todas las máquinas virtuales.
  - Detención y apertura de máquinas virtuales arrancadas independientemente del usuario que las arrancó.
  - Edición de imágenes existentes.
  - Gestión de usuarios.
  - Gestión de servidores de máquinas virtuales.
  - Gestión de grupos de asignatura, así como de las máquinas virtuales asociadas a cada grupo.

Además de estos tipos de usuarios, *CygnusCloud* también ofrece acceso público a ciertas páginas. Estas páginas podrán ser vistas por cualquier usuarios que se conecte a la web de *CygnusCloud*, aunque no se encuentre registrado.

Como ya hemos dicho, cada uno de estos grupos de primer nivel filtran el conjunto de páginas al cual se quiere acceder.

En el segundo nivel las páginas se encuentran agrupadas por secciones. Entendemos como una sección, el conjunto de páginas que abordan un aspecto común para un tipo de usuario concreto. Cada tipo de usuario tendrá acceso a unas secciones u otras. Algunos ejemplos de secciones son la gestión de usuarios, la interacción con las máquinas virtuales o el inicio de sesión. En el cuadro 4.12 puede verse el conjunto de secciones asociadas a cada tipo de usuario.

Por último, en el tercer nivel, las páginas se encuentran organizadas en subsecciones. Cada subsección es una página que contiene las funcionalidades relacionadas con un aspecto concreto y más limitado que el de la sección que la incluye. Existirán por tanto, tantas subsecciones como páginas creadas en la web. En el cuadro 4.13 pueden verse las diferentes subsecciones asociadas a cada sección.

Tipo de usuario	Secciones
Main	Login, About
Student	RunVM, RunningVM, ShowNotifications
Teacher	RunVM, RunningVM, CreateAndEdit, AssociateSubjects, ShowNotifications
Administrator	RunVM, Servers, Users, Subjects, ShowNotifications

CUADRO 4.12: Relaciones entre tipos de usuario y secciones

Sección	Subsección
Login	-
About	-
RunVM	Run, Stop, Open, Edit
RunningVM	StopVM, OpenVM
ShowNotifications	-
CreateAndEdit	CreateVanillaVM, EditVM
AssociateSubjects	-
Servers	Add_servers, Remove_servers, Servers_state, Stop_system
Users	Add, Remove, Associate_subjects
Subjects	Add, Remove

CUADRO 4.13: Relación entre secciones y subsecciones de la web

La barra de secciones permanente en la web muestra la distribución de las páginas en secciones y subsecciones para el tipo de usuario cuya sesión se encuentra abierta.

De cara a la implementación, es necesario ajustar esta estructura para que siga el modelo definido por `web2py`. `web2py` gestiona sus direcciones con respecto a la estructura de los módulos python que actúan como controladores. El conjunto de controladores presentes en la aplicación web coincide con los tipos de usuarios definidos en el primer nivel. Así disponemos de 5 controladores:

- Uno para las páginas de acceso público.
- Uno para las páginas de los alumnos.
- Uno para las páginas de los profesores.
- Uno para las páginas de los administradores.
- Uno para la página que mantendrá el cliente `VNC`.

Este último controlador no corresponde a ningún tipo concreto de usuario pero se define a parte ya que su uso es común para todos.

Dentro de cada controlador es necesario definir una función python para cada una de las secciones del nivel 2 asociadas a este tipo de usuario. Así todo el código presente en los controladores debe pertenecer a una función, bien una función que defina una determinada sección o bien una función auxiliar utilizada por alguna de las funciones principales.

Por último cada una de las funciones python contendrán un variable que les indique cual de las subsecciones del tercer nivel deben controlar en cada momento.

Con respecto a las vistas, el diseño de los niveles se simplifica bastante, siendo necesario tener un fichero `.html` para cada una de las secciones del segundo nivel. El nombre de estos ficheros `.html` deben seguir la siguiente estructura:

```
<Nombre del controlador>/<nombre de la función>.html
```

A modo de resumen, el esquema 4.104 muestra como *CygnusCloud* estructura sus páginas en estos 3 niveles. Como podemos observar, las rutas de direcciones de las que hablamos en la sección 4.5.10.3 coinciden con la estructura en los 3 niveles.

4.5.10.5. La barra de direcciones

Con el fin de permitir a los usuarios acceder a las diferentes páginas de forma directa sin necesidad de utilizar redirecciones a páginas intermedias, la web dispone de una barra de direcciones permanente que permite al usuario redirigirse a cualquier página dentro de su rango de acción. De esta forma dependiendo del tipo de usuario dispondremos de las siguientes barras:

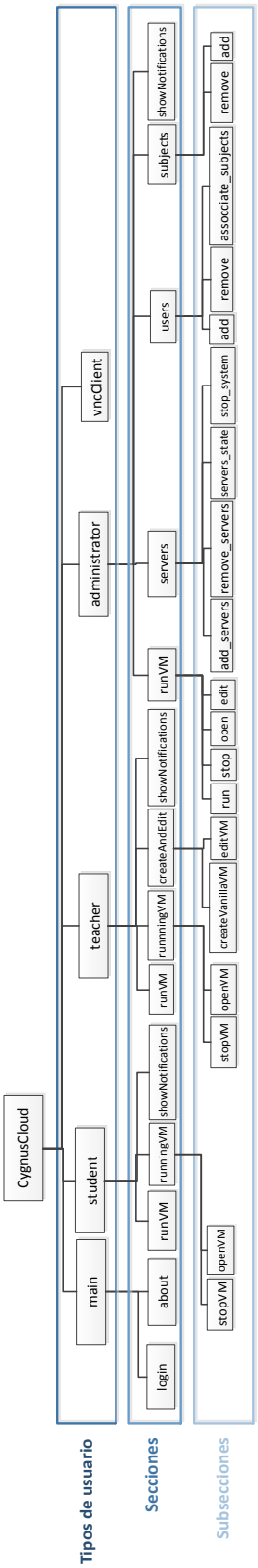

FIGURA 4.104: Diagrama de niveles de la web.

- barra de direcciones para páginas de acceso público. En esta barra aparecen las páginas de inicio de sesión y de `Acerca de`, a las cuales pueden acceder libremente cualquier usuario que entre en la web.
- barra de direcciones para alumnos. Esta barra incluye las páginas de arranque, apertura y detención de máquinas virtuales y está restringida a usuarios de tipo alumno registrados en la web. También incluye la página de visualización de notificaciones pendientes para alumnos.
- barra de direcciones para profesores. Esta barra incluye las páginas de arranque de máquinas virtuales y creación y edición de imágenes, accesibles para los profesores. También incluye la página de visualización de notificaciones pendientes para profesores.
- barra de direcciones para administradores. Esta barra contiene todas las páginas de gestión de máquinas, servidores, asignaturas y usuarios, además de la página de arranque de máquinas virtuales. Su acceso está restringido a usuarios de tipo administrador. También incluye la página de visualización de notificaciones pendientes para administradores.

4.5.10.6. Gestión de usuarios

Para llevar a cabo la gestión de usuarios hemos optado por utilizar una de las librerías que `web2py` nos ofrece. Esta librería, de nombre `auth`, contiene todas las funcionalidades necesarias con respecto a la gestión de usuarios. `Auth` contiene funciones para gestionar la creación de usuarios, los inicios de sesión y las restricciones de acceso a las diferentes páginas. Además esta librería ofrece también funciones que devuelven los formularios de inicio de sesión y cierre de la misma totalmente implementados para ser directamente incorporados en nuestras páginas.[16]

4.5.10.7. Interacción entre páginas

Aunque bien es cierto que cada página debe ser controlada y representada por un controlador y una vista particulares, es posible que requiera de cierta información proveniente de la página a partir de la cual se accedió a ella.

Por esta razón, nuestra aplicación web debe ser capaz de trasmitir argumentos entre las diferentes páginas. Como ya mencionamos en el apartado 4.5.10.4, las diferentes secciones que componen la web son interpretadas por parte del controlador como simples funciones python sin argumentos. Aunque a simple vista lo más lógico pueda parecer que la forma correcta de trasmitir información entre páginas sea definiendo las funciones python con tantos argumentos como datos de entrada deba recibir esto no sucede así. Las razones por las cuales el envío y recepción de datos entre páginas no se da de esta forma son básicamente dos:

1. La redirección de una página a otra no puede darse llamando a la función del controlador directamente. Cuando el desarrollador quiera realizar una redirección, deberá llamar a una función `redirect` con el nombre del controlador que la contiene, el nombre de la función de la sección correspondiente, los argumentos necesarios y el conjunto de variables. Una vez llamado a `redirect`, es `web2py` el encargado de llamar a la función del controlador que corresponda y pasarle los argumentos y variables.
2. El número de argumentos no siempre es el mismo. Dependiendo de la página a partir de la cual pueda realizarse la redirección, es posible que la función de la sección correspondiente reciba más o menos argumentos por lo que estos no pueden definirse como argumentos directos en la definición de la sección.

Por lo tanto la forma correcta de trasmitir información entre las diferentes páginas será utilizando la función `redirect`. Como podemos observar en la definición dada, esta función recibe además del nombre del controlador y la sección, un campo argumentos y un campo variables. Ambos campos permite trasmitir información entre dos páginas con unas características particulares para cada uno de ellos.

En el caso de trasmitir información por el campo de argumentos, esta formará parte de la ruta de direcciones de la página destino. Así todos los argumentos que pasemos por este campo vendrán acoplados a la dirección de la página junto al nombre de la aplicación, el nombre del controlador y el nombre de la sección. Los valores en este campo son trasmitidos como una lista y son accedidos desde la página destino por medio de la llamada a `request.args(i)` siendo `i` la posición de la lista donde se encuentra el elemento deseado.

El segundo campo nos permite transferir la información como variables y no como argumentos. En esta caso el nombre de la variable no formará parte de la dirección destino. Este campo recibe un diccionario con el nombre que se quiere dar a cada una de estas variables y su contenido. Será accedido desde la página destino por medio de la llamada a `request.vars.x` donde `x` es el nombre que se le dio a la variable concreta a la que se quiere acceder.

En el caso de *CygnusCloud*, el campo de argumentos se utiliza para trasmitir el nombre de la subsección que debe ejecutarse dentro de la sección correspondiente. El campo de variables se utiliza para trasmitir el resto de información útil como por ejemplo los resultados de las búsquedas.

4.5.10.8. Seguridad

Con respecto a la seguridad, `web2py`, y por extensión *CygnusCloud*, se centra en resolver los 10 principales problemas de seguridad definidos por OWASP (Proyecto de Seguridad de Aplicaciones Web Abiertas), una comunidad mundial libre enfocada en la mejora de la seguridad de aplicaciones de software[17].

En términos generales, estos 10 problemas son:

- *Cross Site Scripting (XSS)*
- inyecciones *SQL*
- Ejecución de archivos maliciosos
- Referencia directa a objetos
- *Cross Site Request Forgery (CSRF)*
- Fugas de información y manejo de errores inapropiado
- Administración de sesiones y autentificación
- Almacenamiento criptográfico inseguro
- Comunicaciones inseguras
- Restricciones de acceso URL

4.5.10.9. Arranque de una máquina virtual desde la web por un alumno: secuencia básica

En este apartado vamos a ejemplificar de forma detallada cada una de las acciones que debe realizar la aplicación web para atender la petición de arranque de una máquina virtual por parte de un alumno que se conecte a nuestra web.

Una vez que el alumno haya pulsado el vínculo que le da acceso a la página de *CygnusCloud*:

1. Se ejecuta la función `login` dentro del controlador `Main`. Esta función crea el formulario de inicio de sesión (utilizando la utilidad `auth` que se encarga de la gestión de usuarios) y lo devuelve.

2. Justamente después se ejecuta la vista asociada a esta función (`main/login.html`), la cual recibe el formulario creado por el controlador, le aplica el aspecto correspondiente y lo coloca en la página.

3. Una vez el usuario ha introducido su nombre, contraseña y ha pulsado el botón de inicio de sesión, el controlador envía una consulta a la base de datos de la web para comprobar que el usuario existe y que su contraseña es correcta. Tras esto mira el tipo de usuario y redirecciona a la página que sea necesaria. En nuestro caso, al ser un alumno el que inicia la sesión, es redireccionado a la sección `runVM` del controlador `student` con el argumento `run`.

4. Una vez en la función `runVM` del controlador `student`, la cual solo es accesible para alumnos registrados, se evalúa el argumento de entrada para comprobar que subsección debe ejecutarse. Tras esto, se extrae la lista de asignaturas asociadas a este alumno concreto. Para ello, se envía una petición a la base de datos de la web preguntando por estos valores. Para cada una de las asignaturas encontradas se crea una tabla, cuyas filas corresponden a las máquinas virtuales que pueden arrancarse en esta asignatura. La información de las máquinas virtuales asociadas a cada asignaturas se extrae también a partir de consultas a la base de datos, pero en este caso a la del conector. Cada tabla creada se añade al formulario que es devuelto por la función.

5. A continuación se ejecuta la vista `student/runVM.html` la cual recibe el formulario, ajusta su aspecto y lo coloca en la página. Además, esta vista, define un *script* que permite mostrar y ocultar la descripción de las diferentes máquinas virtuales según se encuentren seleccionadas o no.

6. Una vez el usuario ha seleccionado alguna máquina virtual y ha pulsado el botón de arranque, el controlador extrae el identificador de la máquina virtual y pide al conector que arranque dicha máquina. El conector devuelve los parámetros de conexión, los cuales son enviados como variable en la redirección a la página `vncPage`, la cual es creada en una nueva pestaña.

7. Llegados al controlador `vncClient` y a su sección única `vncPage`, se extrae las variables con la información de conexión y se devuelven como resultado de la función.

8. La vista asociada a este controlador (`vncClient/vncPage.html`) recibe los datos de conexión y ejecuta `noVNC`, el cual se encarga de mostrar el escritorio de trabajo por pantalla.

La figura 4.105 muestra un diagrama de secuencia con todo este proceso.

4.5.10.10. Creación de una imagen desde la web por un profesor: secuencia básica

A continuación se explicarán cada una de las acciones que deberá realizar la aplicación web para poder ejecutar una de las funcionalidades más importantes del sistema, la creación de imagen por parte del profesor a partir de imágenes creadas previamente.

Una vez que el profesor haya accedido a la web de *CygnusCloud*:

1. Se ejecuta la secuencia de inicio de sesión por parte del controlador Main de forma similar a como se especificó en los tres primeros puntos del apartado anterior.

2. Una vez iniciada la sesión, al ser el usuario de tipo profesor, es redireccionado a la función runVM, dentro del controlador teacher.

3. El profesor pulsa en la barra de menú sobre la opción `crear nueva máquina`, dentro de la sección `crear y editar`. En este momento la función llama al método redirect pasando como argumentos createAndEdit, como nombre de la función destino, y createVanillaVM, como variable que indica la subsección concreta.

4. En la función createAndEdit, se comprueba el valor de la variable de estado, se crean los formularios asociados y se devuelven como resultado de la función.

5. El fichero `.html` asociado (`teacher/createVanillaVM.html`) recibe los formularios, aplica el aspecto correspondiente y los coloca en la página.

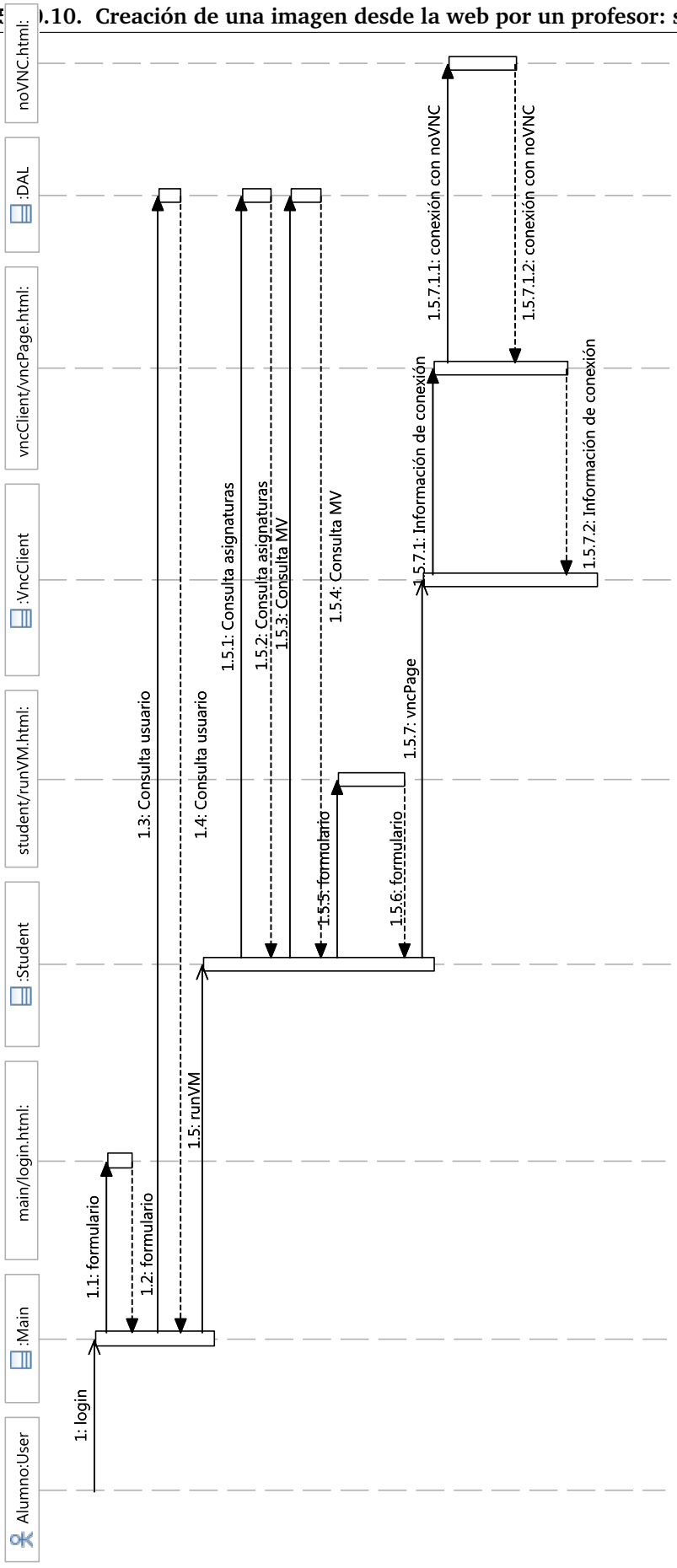

FIGURA 4.105: Secuencia de arranque de una máquina virtual desde la web.

6. En este momento el profesor indica un nombre y una descripción para la imagen que quiere crear. Después selecciona una de las posibles imágenes base que se le ofrecen, teniendo en cuenta el consumo de recursos disponibles en cada una. Es posible filtrar las imágenes por el sistema operativa y la variante que tienen instalados. Una vez rellenado todos los campos y seleccionada la imagen base, se pulsa sobre `crear máquina virtual`.

7. En este momento, la función recopila la información introducida por el profesor, comprueba que no se haya cometido ningún error y llama a la función createImage del conector con esta información.

8. La función createImage envía la petición al sistema y en caso de no producirse ningún error directo, indica a la aplicación web que la petición ha sido enviada.

9. Si todo sale bien en el proceso de creación, el profesor podrá comenzar a editar dicha imagen en la página de edición dentro de la sección `crear y editar`, tal y como se explica en el flujo de ejecución del apartado siguiente.

La figura 4.106 muestra el diagrama de secuencia con el proceso de creación completo.

4.5.10.11. Edición de una imagen en la web: secuencia básica

Además de poder crear nuevas imágenes y asociarlas a un determinado grupo de asignatura, los profesores también pueden editar las imágenes que hubieran creado anteriormente. La edición de una imagen existente se centra en dos aspectos principales :

- Edición del nombre y la descripción asociados a la imagen
- Edición de las herramientas y en general todo el sistema instalado en la imagen

El primer aspecto consiste en una simple modificación en la tabla de la base de datos de la web que asocia el identificador único de una imagen con su nombre y descripción.

Por ello, en este apartado nos centraremos en el segundo punto. El flujo básico necesario para realizar por completo el proceso de edición y desplegado de una imagen se explica a continuación:

1. Se ejecuta la secuencia de inicio de sesión por parte del controlador Main de forma similar a como se especificó en los tres primeros puntos del apartado 4.5.10.9.

2. Una vez iniciada la sesión, se comprueba que el usuario es de tipo profesor y se salta a la función runVM dentro del controlador teacher, con un valor de variable de estado run.

3. El profesor debe pulsar sobre la opción `editar máquina` en la barra del menú, dentro de la sección `crear y editar`.

4. La función runVM llama a la función redirect con createAndEdit como nombre de la sección destino y editVM como valor de subsección.

5. Una vez en la función createAndEdit, se comprueba la variable de subsección y se realiza una llamada a la función getEditedImageIDs del conector, para obtener el identificador de todas las imágenes que se encuentran en edición asociadas a este usuario. Para cada una de las imágenes en edición, se extrae la información correspondiente con el método getImageData del conector. Dependiendo del estado de edición en el que se encuentre la imagen se añade su información a una de las posibles tablas.

6. Una vez recopilada y estructurada en tablas toda la información sobre imágenes en edición, la función createAndEdit devuelve los formularios. En este momento se llama al fichero `.html` asociado a la edición de imágenes por parte del profesor (`teacher/createAndEdit.html`). En esta vista, además de aplicar el aspecto deseado y colocar los formularios en la página, se comprueba que opciones de edición serán accesibles para cada una de las imágenes dependiendo del estado de edición en que se encuentre.

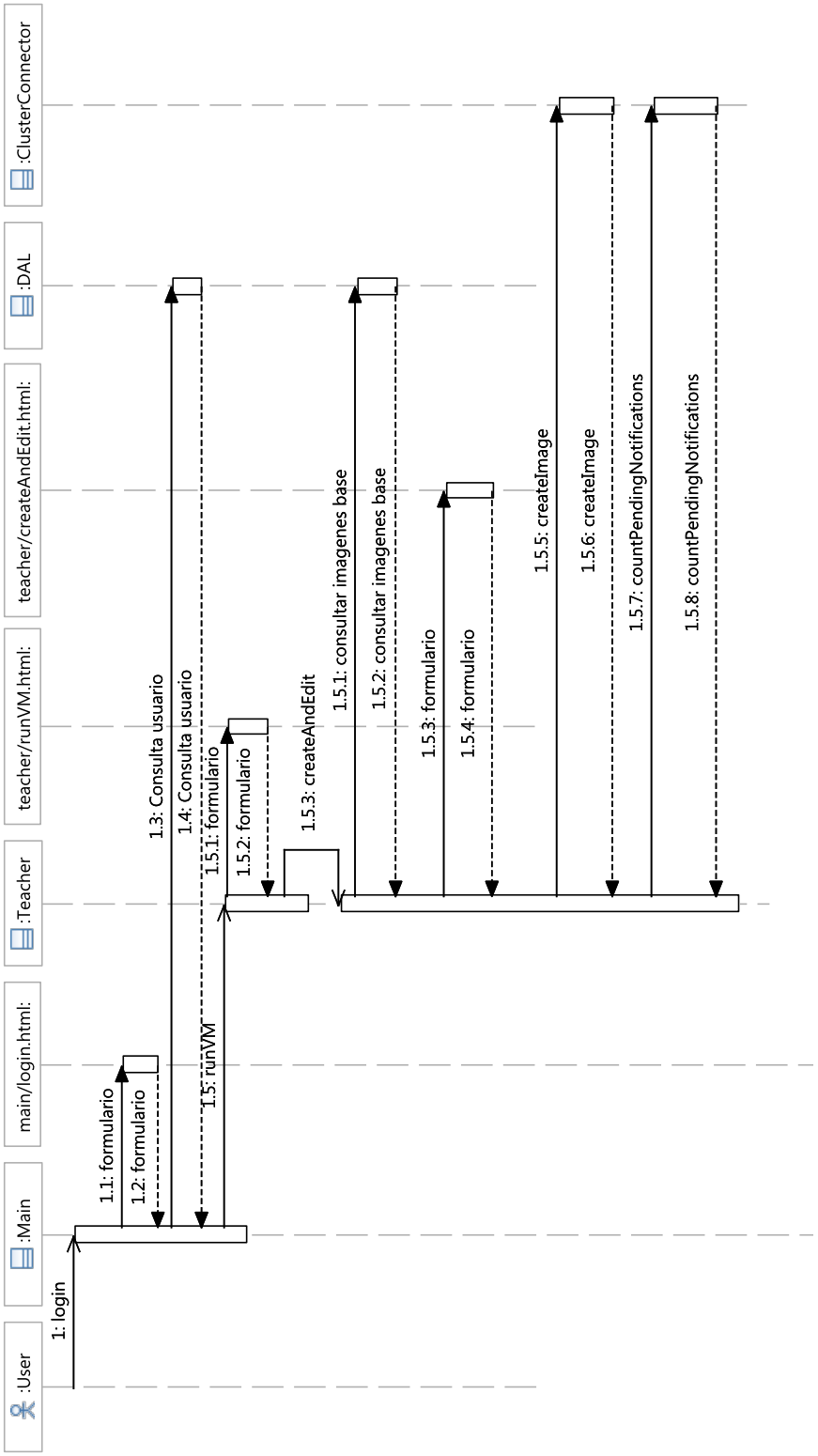

FIGURA 4.106: Diagrama de secuencia creación de una imagen por parte del profesor desde la web

7. En este momento el profesor debe seleccionar la imagen que quiere editar, que se encontrará en un estado parado y sin editar, y pulsar sobre el botón editar.

8. A continuación, la función createAndEdit llama al método editImage del conector y devuelve un mensaje de petición de edición en caso de que no se produzca ningún error directo.

9. En caso de producirse algún error durante el proceso de inicio de edición, se envía el mensaje de error al profesor en forma de notificación y termina el flujo de ejecución. Por otro lado, si todo ha salido bien, se cambia el estado de edición de la imagen, pasando a un estado en edición y ejecutado, y se redirecciona de nuevo a la función createAndEdit.

10. Ahora el profesor debe seleccionar la imagen que quiere editar en la tabla de máquinas en ejecución y pulsar sobre el botón abrir. Una vez pulsado, la función recopila la información de conexión por medio de la llamada a getActiveVMsData y crea una nueva pestaña, la cual pasará a la función vncPage, dentro del controlador vncClient, el cual ejecuta el cliente `noVNC`.

11. Una vez que el profesor ha introducido los cambios deseados en el escritorio remoto y ha apagado la máquina, esta pasa a un estado editada y parada y cambia de tabla en la página de edición de imágenes.

12. Si el profesor vuelve a seleccionar esta imagen, ahora se le muestran las opciones de `seguir editando` y `aplicar cambios`. Como el profesor no quiere introducir ningún cambio más, pulsa sobre el botón `aplicar cambios`.

13. En este momento, la función createAndEdit, comprueba si la imagen ya se encontraba desplegada en el sistema y entonces llama al método desployEditedImage del conector. En caso de que la imagen no se encontrase aun desplegada, ya que acaba de ser creada por el profesor y ha pasado directamente a un estado de edición, entonces se llamará a la función deployNewImage, indicando el número de instancias a crear, el cual depende del número de plazas disponibles en el grupo de asignatura asociada a la imagen en edición.

14. Cuando la imagen se haya desplegado por completo, su estado cambiará a parada y sin editar, y ya podrá ser arrancada por los usuarios asociados.

Como los procesos de subida y bajada de las imágenes en edición pueden ser bastante largos, existen un conjunto de estados de transición intermedios, durante los cuales los usuarios no dispondrán de ninguna opción de edición para las imágenes en edición.

La figura 4.107 muestra el proceso completo de edición de una imagen siguiendo los pasos enumerados.

4.5.10.12. Arranque de un servidor de máquinas virtuales desde la web por un administrador: secuencia básica

Antes de ejecutar una máquina virtual es necesario que al menos uno de los servidores de máquinas virtuales capaz de mantenerla se encuentre arrancado. Los administradores pueden arrancar cualquier servidor de máquinas virtuales que se encuentre registrado en el sistema por medio de la página de edición de servidores.

El flujo básico de arranque de un servidor de máquinas virtuales por parte de un administrador es el siguiente:

1. Se ejecuta la secuencia de inicio de sesión por parte del controlador Main de forma similar a como se especificó en los tres primeros puntos del apartado 4.5.10.9.

2. Una vez iniciada la sesión, se comprueba que el usuario es de tipo administrador y se redirige a la función `runVM` dentro del controlado `administrator`. Esta función crea los formularios de arranque de máquinas virtuales y los devuelve.

3. la vista asociada (`administrator/runVM.html`) recibe los formularios, aplica el aspecto y los coloca en la página.

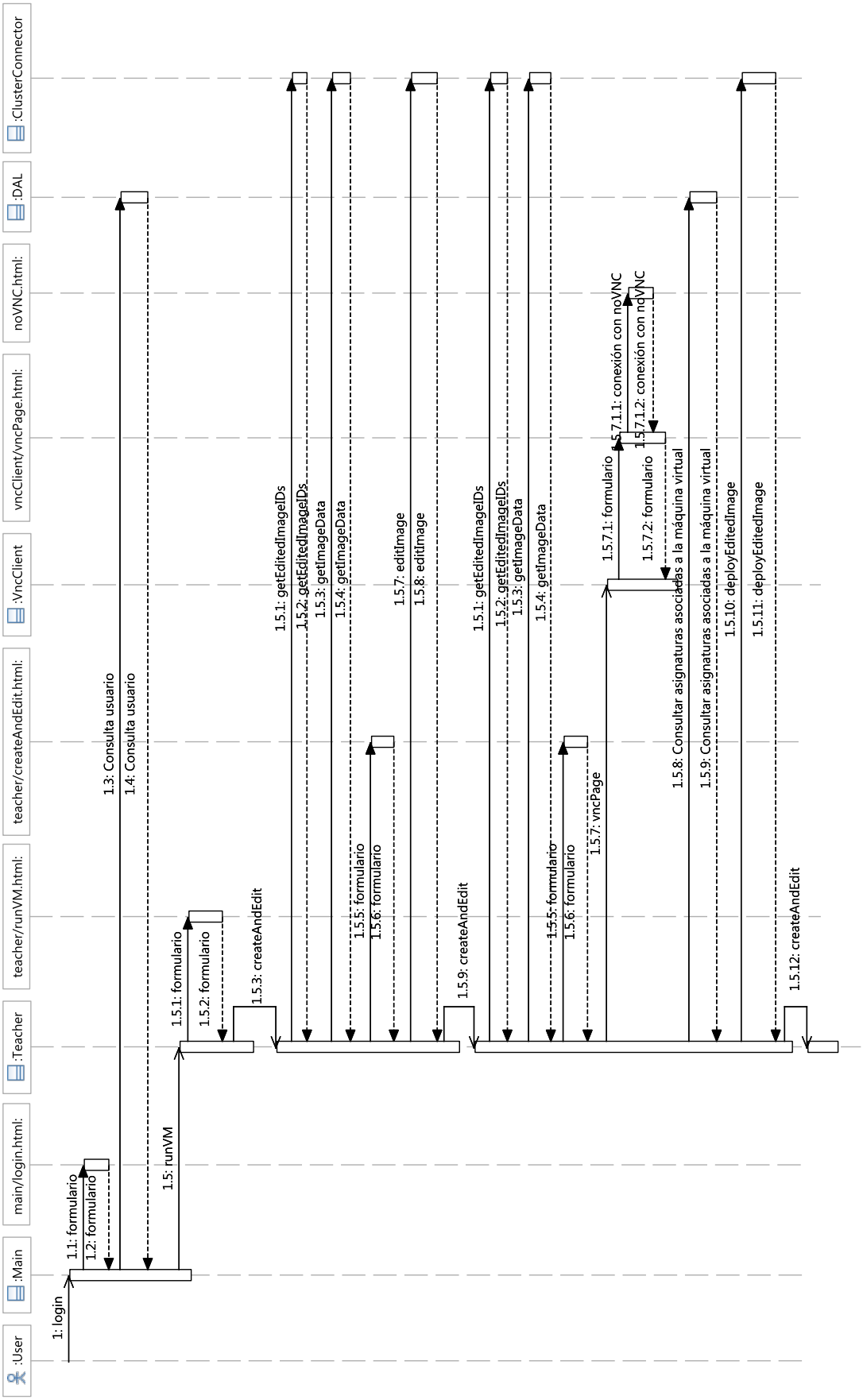

FIGURA 4.107: Diagrama de secuencia de edición de una imagen por parte del profesor desde la web

4. En este momento el administrador pulsa sobre la opción `Editar servidor`, dentro de la sección `Administrar servidores`, en la barra de menú. La función `runVM` aplica la redirección por medio de `redirect`, pasando como nombre de función la sección destino (`servers`) y con una variable de subsección con valor `remove_servers`.

5. Una vez en la función `servers` del controlador `administrator`, se evalúa el valor de la variable de subsección, se pregunta al conector por los servidores actualmente activos y se crea un par de formularios que se devuelven en un diccionario.

6. El fichero `.html` asociado a esta sección (`administrator/servers.html`) recibe el diccionario, extrae los formularios, aplica el aspecto deseado y los coloca en la página.

7. A continuación, el administrador selecciona el servidor que quiere arrancar y pulsa sobre el botón `Buscar servidor`. La función `servers`, consulta al conector el estado del servidor y su información de conexión y llama a la función `redirect` con el mismo valor de función y de variable de subsección anterior pero con una lista adicional con la información encontrada.

8. Al volver a ejecutar la función `servers`, se rellena los formularios con la información de conexión obtenida y se muestran los botones que corresponden al estado del servidor. Como el servidor se encuentra detenido, habrá un botón que permita arrancarlo.

9. Se ejecuta de nuevo la vista `administrator/server.html` con los formularios rellenos, se aplica el aspecto deseado y se coloca en la página.

10. Por último, el administrador pulsa sobre el botón `Arrancar servidor`. La función `servers` envía la petición al conector por medio de la llamada al método `bootUpVMServer` con el nombre del servidor seleccionado y se queda esperando a la respuesta del conector. En caso de que todo haya ido bien, el conector no enviará ningún mensaje de error y el servidor se habrá arrancado.

La figura 4.108 muestra el diagrama de secuencia que representa el arranque de un servidor de máquinas virtuales desde la web.

4.5.10.13. Esquema de la base de datos

La base de datos de la web dispone de las siguientes tablas:

- `auth_user`. Esta tabla contiene información sobre los usuarios registrados en la web. Es utilizada por `auth` para gestionar los inicios y cierres de sesión, además de las restricciones de acceso a cada página. Es creada automáticamente por `web2py` y *CygnusCloud* solo utiliza alguna de sus columnas :
  - `id`. Identificador único asociado al usuario.
  - `email`. Dirección de correo electrónico con la cual el usuario podrá acceder a la web. Por convenio, la estructura de este campo debe seguir el modelo de direcciones utilizada actualmente por la universidad complutense (`<Nombre de usuario>@ucm.es>`). No pueden existir dos usuarios diferentes con el mismo correo electrónico.
  - `password`. Esta columna almacena la contraseña asociada al usuario. Para evitar vulnerabilidades, este campo se almacena codificado en la base de datos y solo se descodifica a la hora de realizar las comprobaciones de inicio de sesión.
- `auth_group`. Esta tabla contiene los diferentes grupos de usuarios disponibles en *CygnusCloud*. Como ya hemos dicho, *CygnusCloud* dispone de tres tipos de usuarios (alumnos, profesores y administradores). Esta tabla es creada automáticamente por `web2py`. Las columnas que la forman son:
  - `id`. Identificador único asociado al grupo de usuarios.
  - `role`. Nombre del grupo.

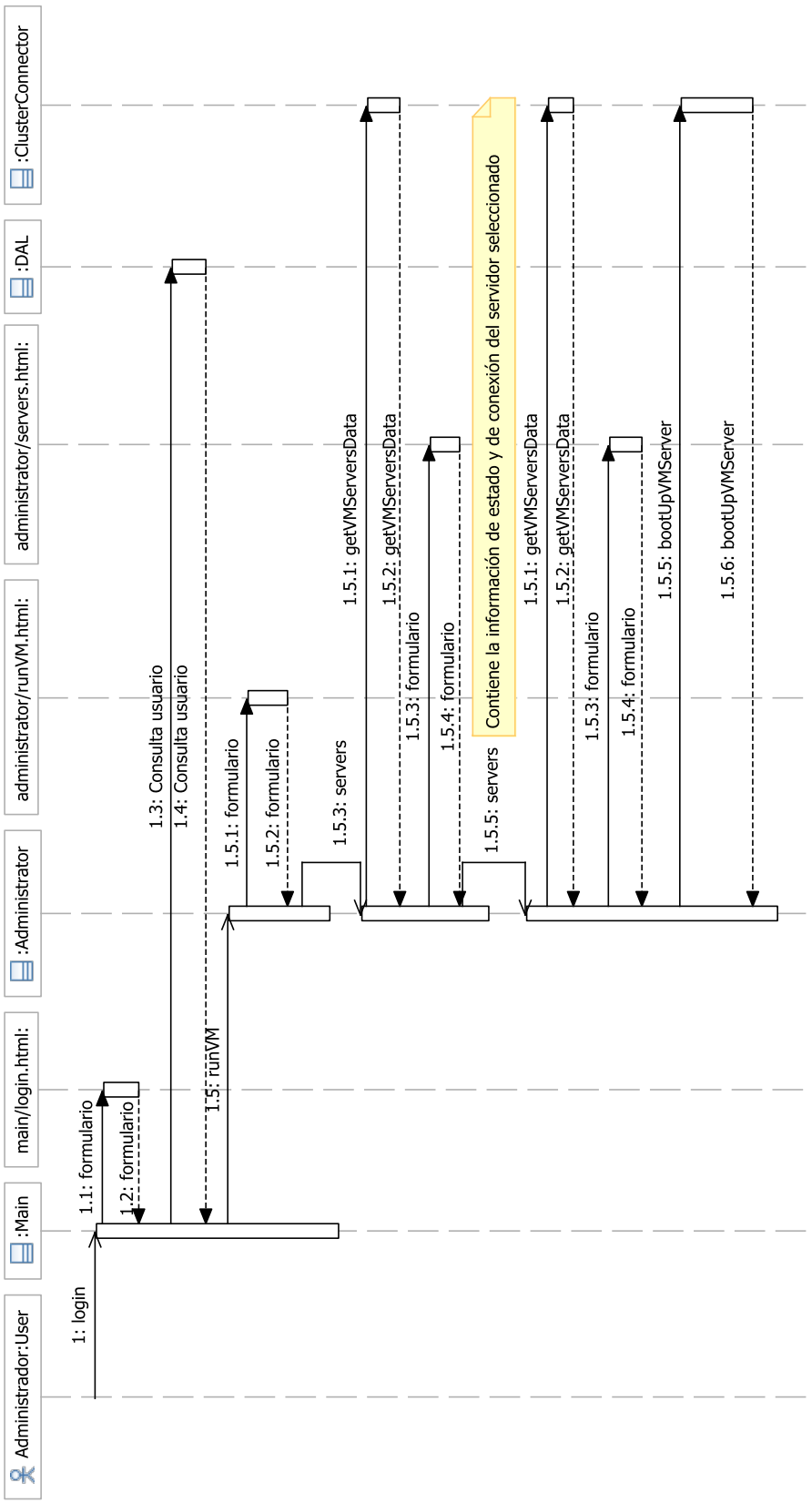

FIGURA 4.108: Diagrama de secuencia de arranque de un servidor de máquinas virtuales desde la web

    - `description`. Breve descripción del grupo de usuarios.

- `auth_membership`. Esta tabla relaciona a cada usuario con el grupo de usuarios al que pertenece. Todos los usuarios deben tener un, y solo un, grupo de usuarios asociados. Al igual que las dos anteriores, esta tabla es creada automáticamente por `web2py` y contiene las siguientes columnas:

    - `id`. Identificador único para la relación concreta.
    - `user_id`. Identificador del usuario registrado en la web.
    - `group_id`. Identificador único del grupo de usuarios al que pertenece el usuario registrado.

- `auth_event`. Esta tabla recopila las últimas acciones que se han llevado a cabo por parte de los usuarios y puede utilizarse para monitorizar el uso que está haciendo cada usuario de la web y mantener un *log* sobre los últimos movimientos. Al igual que las anteriores, esta tabla es creada automática por `web2py` y en este caso, será totalmente gestionada por el módulo `auth`. Las columnas que forman esta tabla son:

    - `id`. Identificador del evento registrado en la base de datos.
    - `time_stamp`. Momento en el cual se llevo a cabo la acción.
    - `client_ip`. dirección ip desde la cual se conectó el cliente.
    - `user_id`. identificador único del usuario que realizó la acción.
    - `origin`. módulo que registro el evento.
    - `description`. Breve descripción sobre la acción registrada.

- `classGroup`. Esta tabla representa los diferentes grupos de asignaturas registrados en el sistema. Cada grupo de asignatura depende no solo de la asignatura que se imparte en ese grupo, sino también del año lectivo en el cual se cursó, y del grupo de clase. Se considera que no existen 2 grupos de asignaturas que impartan la misma asignatura en el mismo grupo de clase. Así esta tabla dispone de las siguientes columnas:

    - `yearGroup`. Año en el cual se cursó dicho grupo de asignatura.
    - `cod`. Código asociado a la asignatura que se imparte en este grupo de asignatura.
    - `curseGroup`. grupo de clase asociado a dicho grupo de asignatura. Por lo general, el grupo de clase viene representado por una letra en minúscula y permite distinguir las diferentes clases que imparten una misma asignatura en un mismo curso lectivo.
    - `placesNumber`. Indica el número de plazas disponibles en este grupo de asignatura. Esta columna se usa para determinar el número de instancias de una imagen que deben crearse a la hora de desplegarse, con el fin de poder ofrecer a todos los usuarios matriculados en este grupo de asignatura una máquina virtual con la que poder trabajar en paralelo con el resto de sus compañeros.

- `userGroup`. Esta tabla relaciona a cada usuario, identificado por su cuenta de correo, con el grupo de asignatura en la que se encuentra matriculado. El grupo de asignatura vendrá determinado por el código de la asignatura y el grupo de clase. Independientemente del tipo de usuario, este se relacionará con los grupos de asignatura en la tabla de forma similar. Las columnas que forman esta tabla son:

    - `userId`. Identificador del usuario. Viene definido por el correo electrónico asociado al usuario.
    - `cod`. Código de la asignatura que compone el grupo de asignatura con el cual se relaciona el usuario determinado.
    - `curseGroup`. Grupo de clase del grupo de asignatura que se relaciona con el usuario.

- `subjects`. Identifica las diferentes asignaturas registradas en la web por medio de un grupo de asignatura. Contiene las siguientes columnas.
  - `code`. Código que representa a la asignatura. Será único para todas las asignaturas registradas.
  - `name`. Nombre de la asignatura asociada al código indicado.
- `VMByGroup`. Esta tabla asocia cada máquina virtual registrada, identificada por su id único, con el grupo de asignatura en la cual se ha registrado la máquina virtual. Un usuario, a excepción de los administradores, solo puede acceder a las máquinas virtuales que se encuentren registradas en uno de los grupos de asignaturas en los cuales se haya matriculado dicho usuario. Las columnas que formas esta tabla son:
  - `VMId`. identificador único para la máquina virtual.
  - `cod`. Código asociado a la asignatura registrada en un grupo de asignatura concreto.
  - curseGroup. Grupo de clase asociado al grupo de asignatura.
- `osPicture`. Define el conjunto de imágenes, registradas en la web, que representan cada una de las variantes de sistemas operativos que pueden estar instalados en las imágenes. Esta tabla contiene las siguientes columnas:
  - `osPictureId`. Identificador único que define la imagen.
  - `picturePath`. Ruta donde se encuentra la imagen asociada a una posible variante de algún sistema operativo.
- `pictureByOSId`. Esta tabla relaciona un sistema operativo y su variante con la imagen que se le da en la web. Así, esta tabla se encuentra compuesta por las siguientes columnas:
  - `osId`. Identificador único del sistema operativo definido.
  - `variantId`. Identificador único de la variante del sistema operativo definido en `osId`.
  - `pictureId`. identificador único que referencia a la imagen con la que se representa la variante del sistema operativo en la web.

4.6. Vista de despliegue

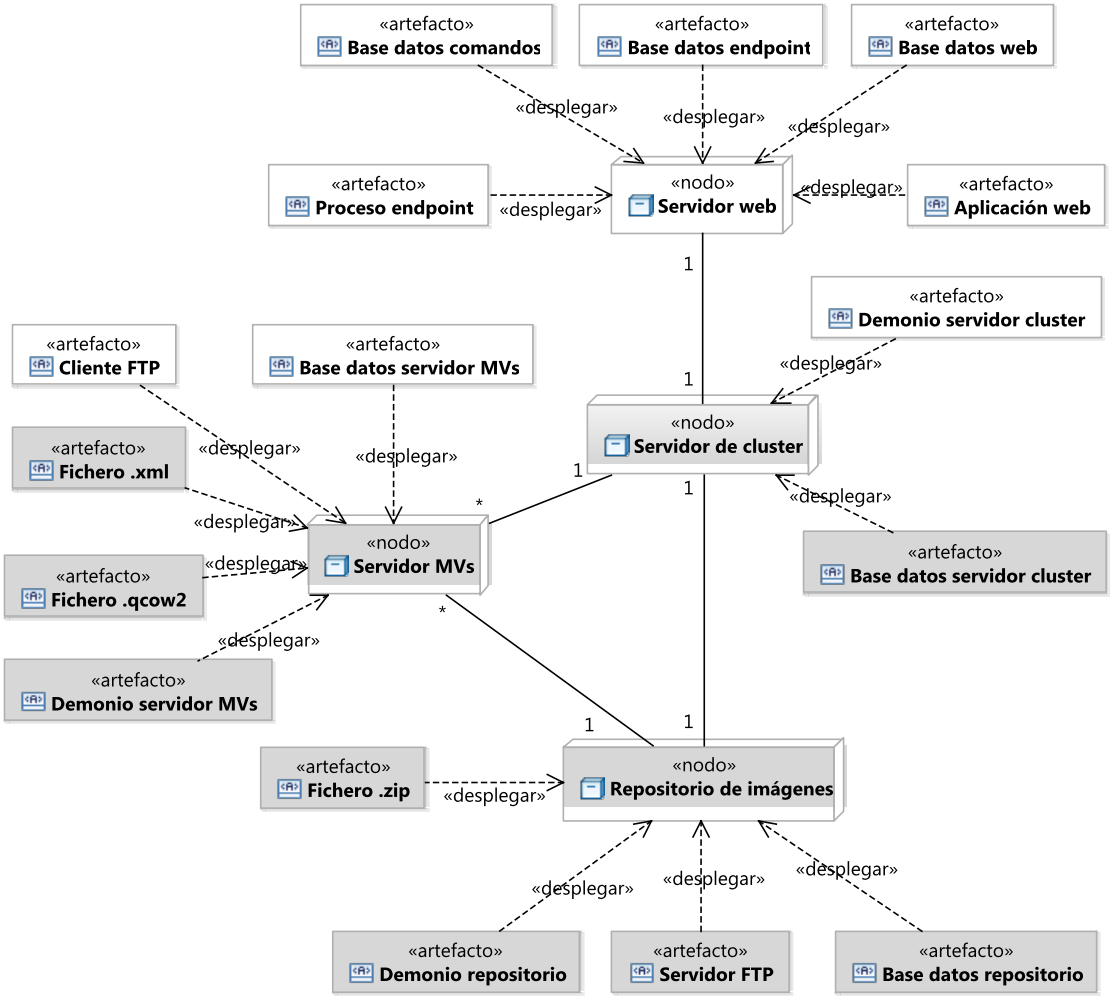

FIGURA 4.109: Diagrama de despliegue de *CygnusCloud*

Tal y como dijimos en la sección 4.4.2, en *CygnusCloud* existen cuatro tipos de servidores:

- los **servidores de máquinas virtuales**. Estos equipos albergarán las máquinas virtuales con las que trabajan los usuarios. Asimismo, también se utilizarán para crear o editar las imágenes o configuraciones que utilizan las máquinas virtuales.
- los **repositorios de imágenes**. Cada uno de estos equipos almacena todas las imágenes o configuraciones que es posible utilizar en cierto *cluster*.
- los **servidores de *cluster***. Estos equipos se utilizan para
  - comprobar si un *cluster* puede atender las peticiones que le llegan,
  - realizar el balanceado de carga entre los distintos servidores de máquinas virtuales de cierto *cluster*, y para
  - procesar las altas, bajas, arranques y apagados de esos servidores de forma centralizada.
- los **servidores web**. Estos equipos albergan una aplicación web, que permitirá a los usuarios interactuar con cierto *cluster* de forma sencilla.

De acuerdo a lo que hemos estudiado en la vista lógica, en un momento dado cierto *cluster* contiene

- 0 o más servidores de máquinas virtuales. Estos interactúan con el servidor de *cluster* y con el repositorio de imágenes.
- un repositorio de imágenes. Estas máquinas interactúan con los servidores de máquinas virtuales y con el servidor de *cluster*.
- un servidor de *cluster*. Esta máquina interactuará con el resto de máquinas del *cluster* y con el servidor web.

Por otra parte, para atender las peticiones de los usuarios, el servidor web se comunicará con el servidor del *cluster*. Todo esto se refleja en el diagrama de despliegue de la figura 4.109.
Como podemos observar en el diagrama de despliegue de la figura 4.109,

- todas las máquinas de la infraestructura albergan una base de datos y un demonio, que procesará las peticiones recibidas a través de una conexión de control.

  La base de datos permitirá que el demonio pueda atender todas esas peticiones de forma adecuada.
- el servidor web alberga tres bases de datos, una aplicación web y un demonio. Como ya sabemos,
  - la aplicación web se comunicará con el demonio utilizando la base de datos de comandos y la base de datos del *endpoint*.
  - la aplicación web utilizará la base de datos restante para, entre otras cosas, realizar el control de acceso.
- el repositorio de imágenes almacena ficheros `.zip` con los datos de las distintas configuraciones. Además, también alberga un servidor FTP, que le permitirá intercambiar ficheros `.zip` con los servidores de máquinas virtuales del *cluster*.
- los servidores de máquinas virtuales albergan un cliente FTP para intercambiar ficheros comprimidos con el repositorio de imágenes.
- los servidores de máquinas virtuales almacenan las imágenes de disco `qcow2` y los ficheros de definición que se usarán para crear máquinas virtuales.

Por claridad, en el diagrama de despliegue no hemos mostrado todos los artefactos de los que dependen los distintos demonios, el proceso *endpoint* y la aplicación web. En la Vista de implementación, estudiaremos estas dependencias en detalle.

4.7. Vista de implementación

Como dijimos en la sección 4.4.2, *CygnusCloud* se descompone en cuatro subsistemas: el servidor de *cluster*, el servidor de máquinas virtuales, el servidor web y el repositorio de imágenes.

En esta sección, mostraremos los artefactos más relevantes que se usan en estos subsistemas. Para ello, distinguiremos cuatro casos: uno para subsistema de *CygnusCloud*.

Por claridad, en esta sección no haremos referencia a los ficheros que, como los ficheros .zip o las imágenes de disco, sólo contienen los datos que se utilizarán para crear las máquinas virtuales. Asimismo, tampoco haremos referencia a las imágenes, hojas de estilo, ficheros JavaScript,... utilizados en la implementación de la aplicación web.

4.7.1. Repositorio de imágenes

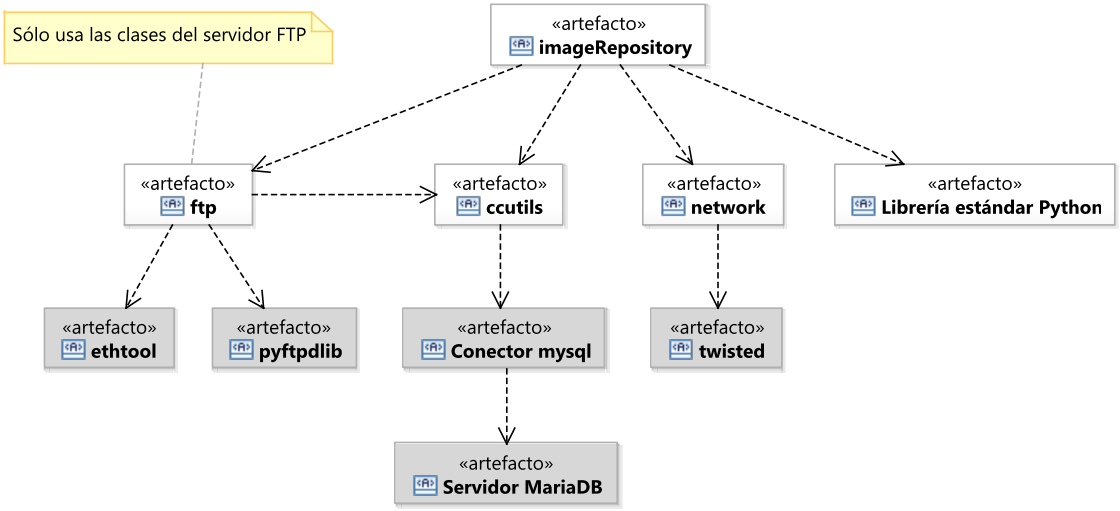

FIGURA 4.110: Principales artefactos utilizados en el repositorio de imágenes

Como puede observarse en el diagrama de la figura 4.110, en el repositorio de imágenes se utilizan diez artefactos principales:

- los módulos del servidor FTP `pyftpdlib`. Estos contienen el código básico del servidor FTP que utiliza el repositorio de imágenes.
- los módulos de la librería de red *twisted*. Estos se utilizarán para manipular los *sockets* asociados a las distintas conexiones de red.
- el ejecutable `ethtool`. Este se utilizará para averiguar el ancho de banda que puede utilizar el tráfico FTP.
- los módulos servidor del paquete `ftp`. Estos se utilizarán para crear y utilizar el servidor FTP a un elevado nivel de abstracción.
- los módulos de la librería estándar de *Python*. Estos se distribuyen junto con el intérprete de *Python*, y estarán disponibles siempre que este esté instalado.
- el ejecutable del gestor de bases de datos MariaDB. Este se utilizará para manipular la base de datos del repositorio de imágenes.
- los módulos del conector oficial *Python* de MySQL. Estos se utilizarán para interactuar con el gestor de bases de datos MariaDB desde código *Python*.

- los módulos de los paquetes `network` y `ccutils`. Estos se utilizan para realizar las comunicaciones con el resto de máquinas de la infraestructura, y también en la implementación del demonio del repositorio de imágenes.
- los módulos del paquete `imageRepository`. Como sabemos, estos módulos contienen todo el código específico del demonio del repositorio de imágenes y su punto de entrada.

Por otra parte, los módulos de los paquetes `network` y `ccutils`, los módulos de la librería de red *twisted*, el ejecutable del gestor de bases de datos MariaDB y los módulos del conector de MySQL estarán presentes en el resto de máquinas de la infraestructura y en el servidor web. De ahora en adelante, hablaremos brevemente de ellos.

Finalmente, es importante notar que sólo los módulos de los paquetes `network`, `ftp`, `ccutils` e `imageRepository` forman parte de la distribución de *CygnusCloud*. El resto de artefactos utilizados en el repositorio de imágenes deberán instalarse manualmente.

4.7.2. Servidor de máquinas virtuales

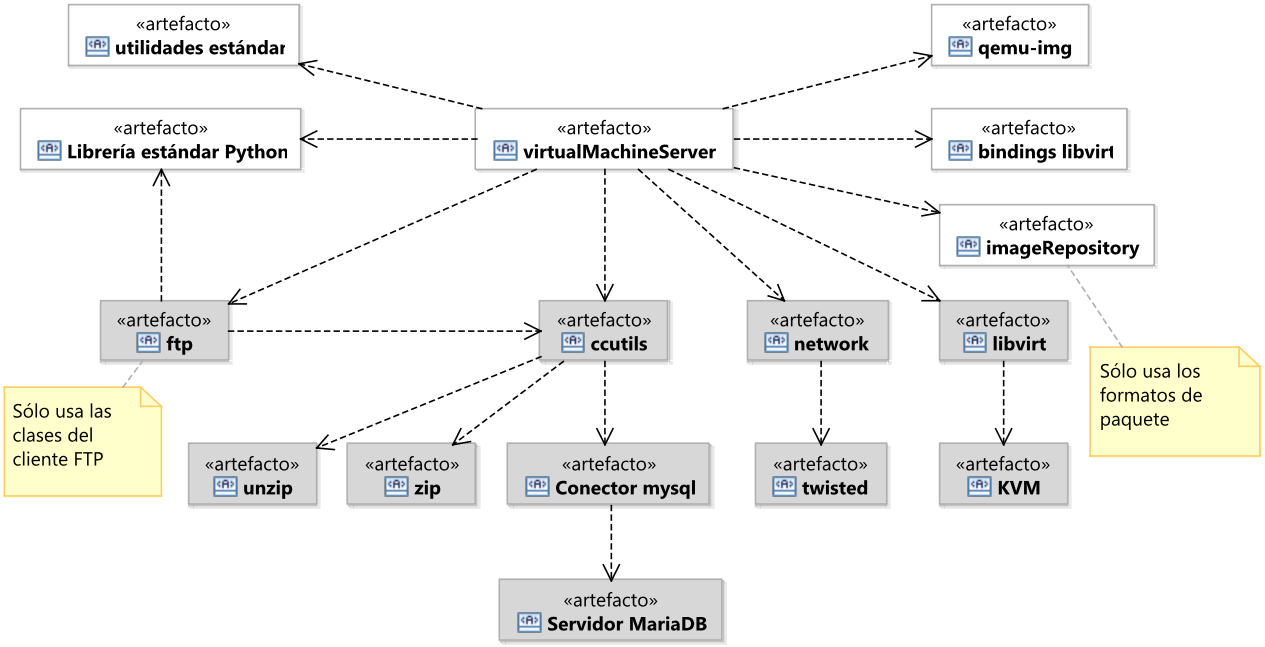

FIGURA 4.111: Principales artefactos utilizados en el servidor de máquinas virtuales

Los principales artefactos que se utilizan en el servidor de máquinas virtuales son los siguientes:

- los ejecutables `zip` y `unzip`, que se utilizan para crear y extraer los ficheros .zip que se intercambiarán con el repositorio de imágenes.
- el ejecutable `qemu-img`, que se utiliza para crear las imágenes de disco *copy-on-write*.
- los ejecutables `cp` y `rm`, que forman parte de las utilidades estándar de los sistemas tipo UNIX.
- los módulos de la librería de red *twisted*.
- los módulos cliente del paquete `ftp`. Estos se utilizarán para utilizar el cliente FTP de la librería estándar de *Python* a un elevado nivel de abstracción.
- los módulos de la librería estándar de *Python*.

- el ejecutable del gestor de bases de datos MariaDB.
- los módulos del conector oficial *Python* de MySQL.
- los módulos de los paquetes `network` y `ccutils`.
- los módulos del paquete `imageRepository` que manipulan los distintos tipos de paquete del repositorio de imágenes.

  Estos módulos se utilizarán para leer y generar los paquetes que los servidores de máquinas virtuales intercambiarán con el repositorio de imágenes.
- el hipervisor KVM, que se utilizará para gestionar las máquinas virtuales que se alojan en el servidor.
- la librería `libvirt`, que se utilizará para interactuar con el hipervisor KVM.
- los *bindings Python* de `libvirt`. Estos se utilizarán para interactuar con la librería `libvirt` desde código *Python*.
- los módulos del paquete `virtualMachineServer`. Como sabemos, estos módulos contienen todo el código específico del demonio del servidor de máquinas virtuales y su punto de entrada.

Finalmente, es importante notar que sólo los módulos de los paquetes `network`, `ftp`, `ccutils`, `imageRepository` y `virtualMachineServer` forman parte de la distribución de *CygnusCloud*. El resto de artefactos utilizados en el servidor de máquinas virtuales deberán instalarse manualmente.

4.7.3. Servidor de *cluster*

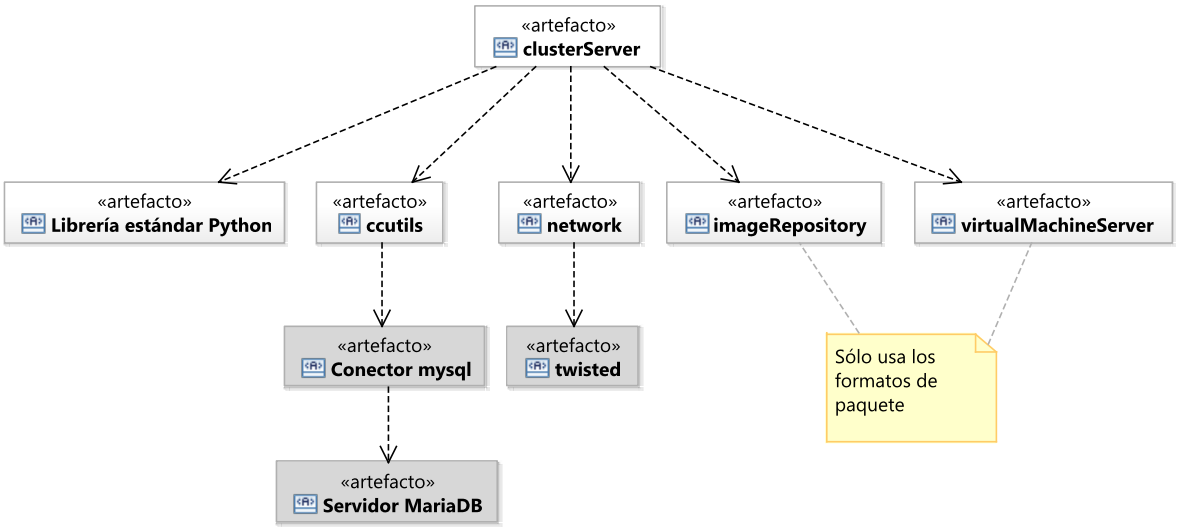

FIGURA 4.112: Principales artefactos utilizados en el servidor de *cluster*

Los principales artefactos que se utilizan en el servidor de *cluster* son los siguientes:

- los módulos de la librería de red *twisted*.
- los módulos de la librería estándar de *Python*.
- el ejecutable del gestor de bases de datos MariaDB.
- los módulos del conector oficial *Python* de MySQL.

- los módulos de los paquetes `network` y `ccutils`.
- los módulos del paquete `imageRepository` que manipulan los distintos tipos de paquete del repositorio de imágenes.

  Estos módulos se utilizarán para leer y generar los paquetes que los servidores de *cluster* intercambiarán con el repositorio de imágenes.
- los módulos del paquete `virtualMachineServer` que manipulan los distintos tipos de paquete del servidor de máquinas virtuales.

  Nuevamente, estos módulos se utilizarán para leer y generar los paquetes que los servidores de *cluster* intercambiarán con los los servidores de máquinas virtuales.
- los módulos del paquete `clusterServer`. Como sabemos, estos módulos contienen todo el código específico del demonio del servidor de *cluster* y su punto de entrada.

Finalmente, es importante notar que sólo los módulos de los paquetes `network`, `ccutils`, `imageRepository`, `virtualMachineServer` y `clusterServer` forman parte de la distribución de *CygnusCloud*. El resto de artefactos utilizados en el servidor de cluster deberán instalarse manualmente.

4.7.4. Servidor web

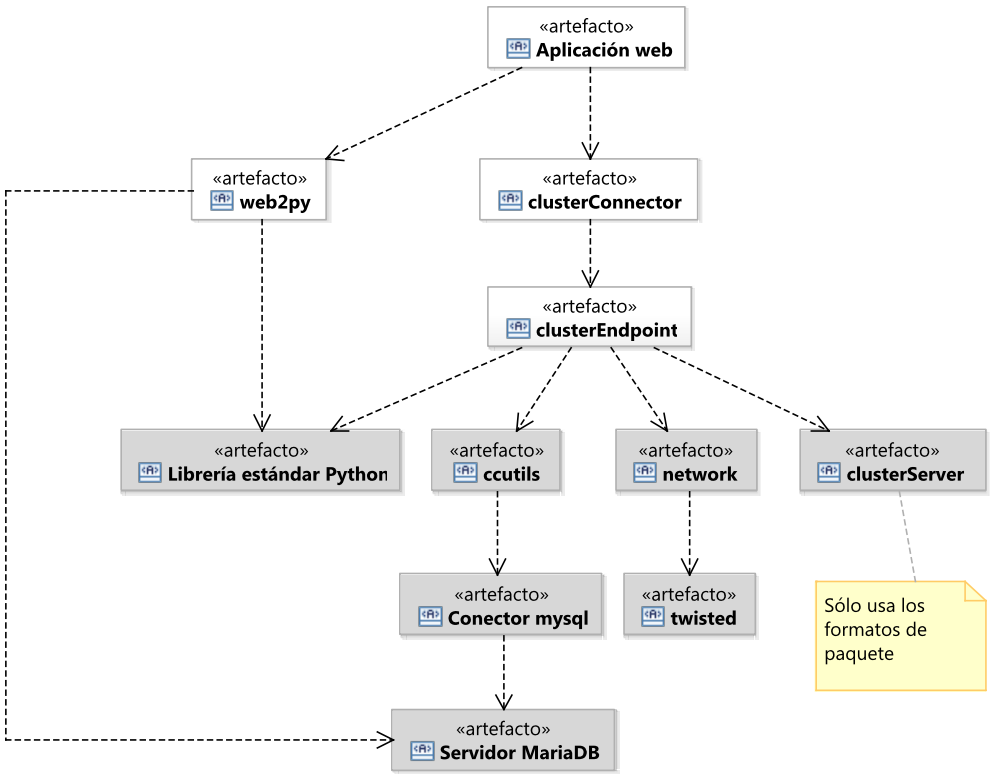

FIGURA 4.113: Principales artefactos utilizados en el servidor web

Los principales artefactos que se utilizan en el servidor web son los siguientes:

- los módulos de la librería de red *twisted*.
- los módulos de la librería estándar de *Python*.

- el ejecutable del gestor de bases de datos MariaDB.
- los módulos del conector oficial *Python* de MySQL.
- los módulos de los paquetes `network` y `ccutils`.
- los módulos del paquete `clusterServer` que manipulan los distintos tipos de paquete del servidor de *cluster*.

  Estos módulos se utilizarán para leer y generar los paquetes que los procesos *endpoint* intercambiarán con los servidores de *cluster*.
- los módulos del paquete `clusterEndpoint`. Como sabemos, estos módulos contienen todo el código específico del proceso *endpoint* y su punto de entrada.
- los módulos del paquete `clusterConnector`. Como sabemos, estos permiten que la aplicación web se comunique con el proceso *endpoint* a través de una base de datos.
- los módulos del *framework* web `web2py`. Estos se utilizarán como base para construir la aplicación web, y también para arrancar el servidor web a través del cual se interactuará con ella.
- los módulos de la aplicación web de *CygnusCloud*.

Finalmente, es importante notar que sólo los módulos de la aplicación web, junto con los módulos del *framework* `web2py` y los módulos de los paquetes `network`, `ccutils`, `clusterServer` y `clusterEndpoint`, forman parte de la distribución de *CygnusCloud*. El resto de artefactos utilizados en el servidor web deberán instalarse manualmente.

Capítulo 5

Estudio de costes

5.1. Introducción

Desde el principio del desarrollo del proyecto pensamos en reutilizar los recursos de los que disponíamos, para así poder reducir el coste total lo máximo posible. Como *CygnusCloud* utiliza un *cloud* privado, esto nos permite decidir que computadores componen dicho *cloud*. Así, podemos valorar libremente que requisitos necesitan los distintos componentes de *nuestro sistema* para saber que recursos dedicar a cada propósito.

Aunque la reutilización de recursos es una opción, también podemos dedicar *hardware* específico para tratar la carga de trabajo que tendrán los distintos componentes. Para ello hemos creado tres configuraciones de servidores, cada una con unas características mejores que la anterior, con las que discutiremos que opción es la mejor para cada componente. Estas configuraciones se pueden ver en el cuadro 5.1.

5.2. Requisitos mínimos

Los siguientes requisitos mínimos están pensados para ser utilizados en PCs.

5.2.1. Servidores de máquinas virtuales.

Para los servidores de máquinas virtuales necesitamos que los procesadores permitan usar virtualización, esto no es un gran requisito, ya que tanto AMD (desde 2006) como Intel (desde 2005), incluyen en sus procesadores tecnología que soporta virtualización completa. Por ello, en la actualidad, casi cualquier ordenador tiene este tipo de tecnología incorporado. Esto nos permite usar ordenadores como servidores, aunque debido a las limitaciones, tanto de CPU como de memoria RAM, nos impone un número máximo de máquinas que es capaz de soportar.

Si nos fijamos en los cuadros 4.1 y 4.2, y las características de los PCs actuales (un mínimo de dos núcleos, 4 GB de RAM y 200 GB de disco duro), fácilmente podríamos arrancar dos máquinas *small* o una *medium*.

5.2.2. Servidor de *cluster*

El servidor de *cluster* tiene que soportar mucha carga de trabajo, ya que debe comunicarse con el resto de componentes y gestionarlo todo. Para poder hacer su trabajo se necesita como mínimo un

	CPUs	**RAM**	**Disco Duro**	**Precio**	***# small***	***# medium***	***# big***
pequeño	2	4	500	600 €	2	1	-
medio	4	4	1000	850 €	4	2	1
grande	6	8	1000	1250 €	6	3	1

CUADRO 5.1: Precios servidores

Máquina	CPU	RAM	Disco Duro
Servidor de *cluster*	2	4 GB	-
Servidor web	2	2 GB	100 GB
Repositorio	1	2 GB	500 GB
Cliente	1	1 GB	-

CUADRO 5.2: Requisitos mínimos para soportar la infraestructura de *CygnusCloud*

PC con 2 *cores* y 2 GB de RAM. En cuanto al espacio en disco no requiere demasiado, ya que su trabajo consiste básicamente en comunicar al resto lo que tiene que hacer.

5.2.3. Servidor web

Aunque normalmente las páginas web deben estar preparadas para atender a cientos de miles, incluso millones, de usuarios, esta web solo va a ser de uso interno a la institución, lo que supondría a lo sumo varios miles de usuarios conectados a la vez. Esto nos permite usar un PC con un par de núcleos, 2 GB de RAM y 100 GB de disco. Con esto, podríamos gestionar toda la información, tanto de los usuarios como de las máquinas que se pueden arrancar.

5.2.4. Repositorio

El repositorio no requiere gran cantidad de recursos, salvo por el almacenamiento. Esto se debe a que se comporta básicamente como un servidor FTP pero con un número limitado de conexiones, que como se ha dicho en la sección 4.5.4.2, es configurable. En ningún caso debería admitir muchas mas conexiones de 10, ya que puede provocar un cuello de botella.

Con respecto al tiempo de CPU no será muy elevado, ya que la mayor parte del tiempo estará enviando o recibiendo archivos.

Con respecto a la memoria RAM, debería bastar con 2GB como máximo, los cuales se usarían principalmente en la cache de archivos. Esto es una ventaja ya que se podría usar cualquier ordenador, incluso un ordenador de bajo consumo, aprovechando el bajo coste de los mismos. Lo que sí que es importante, es el almacenamiento, ya que es donde se guardarán todas las máquinas virtuales.

5.2.5. Resumen

Esto requisitos mínimos se pueden cubrir fácilmente con PCs por lo que el coste sería cero.

Todos estos requisitos mínimos aparecen resumidos en el cuadro 5.2.

Como se puede observar, en el cuadro no aparecen los servidores de máquinas virtuales, esto se debe a que sus requisitos mínimos dependen del tipo de máquina que se quiera ejecutar.

5.3. Servidor de máquinas virtuales

En los servidores de máquinas virtuales, el tipo de servidores que necesitamos depende del número máximo de máquinas virtuales que queramos tener activas en un momento dado.

Dado los distintos tipos de máquinas virtuales que podemos tener (4.1 y 4.2) y el uso que se van a hacer de estas, una posible distribución del uso de cada tipo podría ser la que se propone en el cuadro 5.3.

Tipo de máquina	Presencia (%)
small	60
medium	30
big	10

CUADRO 5.3: Distribución de los distintos tipos de máquinas

En una institución como la `Facultad de Informática` de la `Universidad Complutense de Madrid`, existen unos 300 puestos de laboratorios. Estableciendo esto como tope y suponiendo que los distintos tipos de máquinas que existen se distribuyen como se indica en el cuadro 5.3, es necesario un sistema capaz de soportar lo siguiente:

$$\text{Máquinas } \textit{small}\text{: } 300 \cdot 0,6 = 180$$

$$\text{Máquinas } \textit{medium}\text{: } 300 \cdot 0,3 = 90$$

$$\text{Máquinas } \textit{big}\text{: } 300 \cdot 0,1 = 30$$

Teniendo en cuenta esto y los servidores de máquinas virtuales propuestos al principio del capítulo 5.1 veremos cuanto puede costar una infraestructura que soporte esas 300 máquinas:

- Para poder soportar las 30 máquinas *big*, podríamos usar tanto un servidor *grande* como uno *medio*. La ventaja que tendría usar un servidor *grande* es que también podría albergar una máquina *medium*, por lo que con cada servidor *grande* podemos tener funcionando una máquina *big* y otra *medium*, por un coste de $30 \cdot 1250 = 37500$ €.

- Para las máquinas *medium*, podríamos usar cualquiera de los tres servidores. Como podemos usar el espacio de los servidores *grandes* para almacenar 30 de las máquinas, vamos a hacer los cálculos con las 60 restantes:

  - Con servidores *pequeños*:$60 \cdot 600 = 36000$ €
  - Con servidores *medios*: $\frac{60}{2} \cdot 850 = 25500$ €
  - Con servidores *grandes*: $\frac{60}{3} \cdot 1250 = 25000$ €

  Como podemos ver, parece obvio que la opción de usar servidores *pequeños* está totalmente descartada. Entre las otras dos opciones la diferencia no es muy relevante, por lo habría que valorar si el ahorro en los servidores grandes compensa frente a tener mas máquinas en el mismo servidor físico. La ventaja de esto último es que si surge algún problema en uno de los servidores, estaríamos perdiendo una menor capacidad de procesamiento.

- Para las *small* también es posible utilizar los tres tipos de servidores como opción:

  - Con servidores *pequeños*:$\frac{180}{2} \cdot 600 = 54000$ €
  - Con servidores *medios*: $\frac{180}{4} \cdot 850 = 38250$ €
  - Con servidores *grandes*: $\frac{180}{6} \cdot 1250 = 37500$ €

  En este caso ocurre como antes, la diferencia entre los *medios* y *grandes* no es muy notoria, y tendremos que tener en cuenta la misma consideración aplicada anteriormente.

En resumen, podríamos tener los 300 puestos funcionando por un total de 100000 €.

5.4. Servidor de *cluster*

Como ya se ha comentado en la sección de requisitos mínimos (sección 5.2.2) y aunque las necesidades de RAM y CPU se cubren con un servidor *medio*, quizás la capacidad del disco duro pueda resultar excesiva, ya que la mayoría de los datos que maneja en ejecución, no son guardados entre varias ejecuciones.

5.5. Servidor web y repositorio

Debido a la distinta naturaleza de trabajo del servidor web y el repositorio, ambos podrían compartir un mismo servidor, reduciéndose así el coste total. Los requisitos del repositorio requieren una mayor capacidad de disco y RAM, mientras que el servidor web necesita un mayor tiempo de CPU y tamaño de RAM. El cuadro 5.1 nos indica que podríamos usar un servidor de tamaño *medio* para mantener ambos sistemas.

5.6. Clientes

Los clientes, al usar un cliente VNC web, lo único que requieren es que poder ejecutar un navegador con soporte para las tecnologías HTML, *websockets* y *canvas*, lo cual puede conseguirse con un ordenador de gama baja. Esto supone unos 400 € de coste por equipo.

5.6.1. Renovación de un aula de informático

El hecho de tener ordenadores de gama baja como clientes, nos supone un ahorro importante cuando queramos aplicar el sistema en los ordenadores de todo un aula. Como pudimos ver en el apartado 1.3.3, un ordenador actualmente cuesta entre 784 y 960 €, lo que implica un ahorro mínimo del 49 %.
Teniendo en cuenta esto, vamos a calcular el ahorro que se produciría en cada aula:

- Aula de programación:
  - Coste del equipo: 784 €
  - Coste del cliente: 400 €
  - Coste total del aula sin usar *CygnusCloud*: $784 \cdot 20 = 15680$ €
  - Coste total del aula usando *CygnusCloud*: $400 \cdot 20 = 8000$ €
  - Ahorro: $15680 - 8000 = 7680$ € (49 %)
- Aula de informática gráfica:
  - Coste del equipo: 960 €
  - Coste del cliente: 400 €
  - Coste total del aula sin usar *CygnusCloud*: $960 \cdot 20 = 19200$ €
  - Coste total del aula usando *CygnusCloud*: $400 \cdot 20 = 8000$ €
  - Ahorro: $19200 - 8000 = 11200$ € (58 %)
- Aula de diseño de circuitos:
  - Coste del equipo: 941 €
  - Coste del cliente: 400 €
  - Coste total del aula sin usar *CygnusCloud*: $941 \cdot 20 = 18820$ €
  - Coste total del aula usando CygnusCloud: $400 * 20 = 8000$ €
  - Ahorro: $18820 - 8000 = 10820$ € (57 %)

5.7. Licencias

Aunque actualmente ya se dispongan de licencias de *Windows*, esto no es suficiente. Es necesario tener licencias con derecho a virtualización, lo cual supone un coste de 100 dólares americanos (unos 75 €) anuales por cada 4 máquinas virtuales arrancadas. Por ello, tendremos que decidir el máximo número de máquinas con *Windows* que queremos tener activas.
Siguiendo con los ejemplos del apartado 5.3 y suponiendo que un 60 % de las máquinas están usando *Windows,* podemos concluir que:
$300 \cdot 0{,}6 = 180$ máquinas usan *Windows* lo que supone que necesitamos $180/4 = 45$ suscripciones, y eso se traduce en un coste de $45 \cdot 75 = 3375$ € anuales.

Máquina	Tipo de servidor	Precio
Servidor de *cluster*	*medio*	850 €
Servidor web/repositorio	*medio*	850 €
Cliente	-	400 €

CUADRO 5.4: Servidor para cada componente del sistema

5.8. Coste final

Ahora que ya sabemos cuales serían los mejores servidores para todos los elementos del sistema,podemos concluir en que los costes requeridos en cada uno de ellos son los expresados en el cuadro 5.4.

En el caso de los servidores de máquinas virtuales, el coste final dependerá del número total de puesto y del tipo de máquinas que se quieran tener activas. Si usamos las máquinas del ejemplo anterior (5.3), tenemos que para soportar los 300 puesto de laboratorios necesitaríamos servidores por valor de 100000 €. A esto habría que añadir los servidores necesarios para el resto de la infraestructura (1750 €). Si se quiere hacer una implantación desde cero, también sería necesario proporcionar el coste de los 300 puestos, lo que resultaría:

$$300 \cdot 400 \text{ €} = 120000 \text{ €}$$

Con todo esto, el coste final para poder implantar toda la infraestructura sería de

$$100000 + 1750 + 120000 = 221750 \text{ €}$$

Capítulo 6

Conclusiones

A lo largo del desarrollo de *CygnusCloud* hemos descubierto una rama del mundo de la informática, la del *Cloud Computing*, que nos era totalmente desconocida.

Además, durante los diez meses que ha durado el proceso de desarrollo, también hemos logrado dominar varios lenguajes y tecnologías que, desde nuestro punto de vista, son esenciales para completar nuestra formación y mejorar nuestras perspectivas ante nuestra inminente incorporación al mercado laboral.

Por otra parte, hemos obtenido una visión muy completa del estado del arte actual de todas las tecnologías que hemos utilizado. Así, nos sentimos lo suficientemente capacitados como para asesorar en el uso de las mismas y para sugerir alternativas.

Para comenzar, en esta sección mostraremos los conocimientos que hemos adquirido. Posteriormente, enumeraremos las principales características de *CygnusCloud* y su potencial. Para finalizar, hablaremos de los principales problemas a los que nos hemos enfrentado durante el proceso de desarrollo y de las repercusiones que ha tenido nuestro proyecto.

6.1. Conocimientos adquiridos

6.1.1. *Cloud Computing*

En los últimos años, el grado de adopción de tecnologías *cloud* no ha parado de aumentar. Aunque no lo notemos, la gran mayoría de servicios *web* que utilizamos en nuestra vida diaria, desde las redes sociales hasta los servicios de almacenamiento de ficheros, utilizan, de una manera u otra, una o más tecnologías *cloud*.

Asimismo, considerando la actual coyuntura económica, las tecnologías *cloud* tienen mucho que aportar en lo que a la reducción de costes se refiere, ya que

- permiten eliminar los costes de adquisición y mantenimiento de la infraestructura, que asumirá el proveedor del servicio.
- es posible utilizar el *software* a través de un modelo de pago por suscripción, lo que permite no pagar por las licencias de *software* que no se utilizan.
- los usuarios pueden utilizar equipos menos potentes y, por tanto, más baratos y con un menor consumo energético, ya que todas las tareas de procesamiento tienen lugar en los servidores del proveedor del servicio.

Por otro lado, el estudio de las distintas soluciones basadas en tecnologías *cloud* nos ha permitido aprender cómo atender de forma rápida y eficiente a un gran número de usuarios.

En definitiva, nos parece que el uso de tecnologías *cloud* ha resultado sumamente útil tanto para completar nuestra formación como ingenieros informáticos como para mejorar nuestras perspectivas laborales.

6.1.2. Sistemas IaaS (*Infrastructure as a Service*)

CygnusCloud ofrece un servicio del tipo Infraestructura como Servicio (en inglés, *Infrastructure as a Service* o IaaS).

Aunque hemos diseñado e implementado una infraestructura *ad-hoc*, muchas de las decisiones que hemos tomado durante el proceso de desarrollo están presentes en soluciones del tipo Infraestructura como Servicio existentes en el mercado, como *OpenStack* y *OpenNebula*.

Así, la base de conocimientos que hemos adquirido durante el desarrollo de *CygnusCloud* nos permitirá reducir el tiempo que tardaremos en familiarizarnos con otras soluciones del tipo IaaS existentes en el mercado.

6.1.3. Tecnologías de virtualización

Al inicio del desarrollo, tuvimos que evaluar distintas tecnologías de virtualización *software* (como VirtualBox y VMWare) y distintas tecnologías de paravirtualización y de virtualización completa (Xen y KVM).

Aunque ya estábamos bastante familiarizados con VirtualBox y VMWare, nunca habíamos utilizado Xen ni KVM. Por ello, decidimos familiarizarnos con estos sistemas y hacer pruebas de integración. De esta manera, en la actualidad conocemos sus ventajas e inconvenientes, cómo utilizarlos e incluso su funcionamiento interno.

Por otra parte, hace nueve meses estábamos acostumbrados a interactuar con los sistemas de virtualización a través de interfaces gráficas o a través de la línea de comandos. Por ello, tuvimos que aprender a utilizar la librería `libvirt`.

Considerando todas las funciones que aporta `libvirt` y, sobre todo, la posibilidad de interactuar con múltiples hipervisores de forma muy similar, creemos que los conocimientos que hemos adquirido nos serán de suma utilidad en el futuro.

6.1.4. Tecnologías de red

Todas las máquinas de la infraestructura deben comunicarse entre sí a través de una red. Por ello, durante el desarrollo de *CygnusCloud* ha sido obligatorio el uso de distintas tecnologías de red.

Al inicio del desarrollo, sabíamos manipular *sockets* para comunicar máquinas, por lo que podíamos comprender el funcionamiento de la gran mayoría de librerías de red. No obstante, debido a sus enormes dimensiones y a la "calidad" de su documentación, el uso de la librería *twisted* ha supuesto todo un reto.

En este momento, conocemos al detalle su arquitectura y su funcionamiento interno. Asimismo, hemos construido sobre ella un módulo de comunicaciones de propósito general, que puede utilizarse en cualquier aplicación *Python*.

Por otra parte, también hemos afianzado conceptos de redes que hemos aprendido durante la carrera, tales como el funcionamiento de los protocolos IP versión 4 e IP versión 6, el funcionamiento del protocolo TCP, el formato de las direcciones MAC...

Asimismo, hemos aprendido a configurar redes en modo NAT, lo cual resulta fundamental para superar la gran limitación del protocolo IP versión 4: la escasez de direcciones.

6.1.5. El lenguaje *Python*

En la actualidad, *Python* es uno de los lenguajes de programación de propósito general más utilizados. Por su potencia y simplicidad, cada vez más organizaciones lo adoptan en sus desarrollos. Asimismo, un gran número de servicios web muy populares contienen código escrito en *Python*.

Durante el proceso de desarrollo, hemos logrado dominar las dos ramas de *Python*, la 2.7.x y la 3.x. Estos conocimientos nos serán extremadamente útiles en el mundo laboral.

6.1.6. Tecnologías web

A nuestro parecer, el gran inconveniente de nuestro plan de estudios es la ausencia de contenidos relacionados con las tecnologías web. Durante los cinco años de la carrera, no existe ninguna

asignatura troncal, obligatoria u optativa en la se enseñe a utilizar alguna tecnología web, a pesar de que estas son cada vez más demandadas por el mercado laboral.

Durante el desarrollo de *CygnusCloud*, nos hemos familiarizado con los lenguajes HTML5, CCS3 y *JavaScript*, y también con los *frameworks* `web2py` y *JQuery*.

Los conocimientos que hemos adquirido nos permitirán evaluar y aprender a utilizar nuevas tecnologías web de forma más rápida y adecuada.

6.1.7. Programación concurrente

Para garantizar la eficiencia en los distintos subsistemas de *CygnusCloud*, resulta imprescindible sacar el máximo partido al *hardware*. Y como la gran mayoría de CPUs actuales son *dual-core* o *quad-core* e incluso soportan *multithreading* simultáneo, para aprovechar el *hardware* resulta imprescindible utilizar varios hilos.

Aunque al inicio del proyecto ya contábamos con conocimientos de programación concurrente, durante el proceso de desarrollo los hemos afianzado y ampliado.

6.1.8. Administración de sistemas *Linux*

Al inicio del proceso de desarrollo, contábamos con cierta experiencia en el uso de sistemas *Linux*. No obstante, durante el desarrollo del proyecto hemos aprendido a

- configurar manualmente interfaces de red, crear interfaces de red virtuales y configurar encaminadores.
- utilizar el sistema de gestión de paquetes `apt` de forma avanzada, incluyendo la resolución de conflictos y la reparación de bases de datos de paquetes corruptas.
- compilar e instalar módulos del *kernel*.
- compilar e instalar librerías.
- modificar el proceso de arranque, indicando los módulos del *kernel* que se instalarán y los servicios que se iniciarán.
- gestionar usuarios y grupos de usuarios.

Teniendo en cuenta que en la actualidad *Linux* es el sistema operativo dominante en el mercado de los servidores, creemos que estos conocimientos nos resultarán muy útiles en el futuro inmediato.

6.2. Características y capacidades del sistema *CygnusCloud*

6.2.1. Funciones básicas

En *CygnusCloud*, existen tres tipos de usuarios: alumnos, profesores y administradores.

En primer lugar, los alumnos pueden

- arrancar una máquina virtual asociada a cualquier asignatura en la que estén matriculados y utilizarla para trabajar.
- forzar el apagado o el reinicio de una de sus máquinas virtuales.
- conectarse y desconectarse del servidor VNC asociado a una de sus máquinas virtuales. Esto les permite, por ejemplo, trasladarse a otro aula de informática para continuar trabajando sin necesidad de apagar la máquina.
- introducir y extraer datos de la máquina virtual a través de internet.

Por otra parte, los profesores pueden realizar las mismas tareas que los alumnos. Asimismo, pueden

- crear y borrar las asociaciones entre sus máquinas virtuales y las asignaturas que imparten, pudiendo compartir máquinas virtuales entre distintas asignaturas cuando lo estimen oportuno.
- crear una configuración nueva para una máquina virtual, tomando como base cualquier configuración existente en el sistema. Este proceso puede hacerse en varias fases.

  Por ejemplo, un profesor puede empezar a crear una configuración a última hora de la tarde, apagar la máquina virtual correspondiente y proseguir el día siguiente.
- editar la configuración de una de sus máquinas virtuales. Nuevamente, este proceso puede hacerse en varias fases.
- establecer, desde el sistema operativo de la máquina virtual, conexiones con servidores CIFS o NFS. Esto permite a los profesores utilizar los datos almacenados en su propio equipo para crear o editar configuraciones.
- desplegar una configuración nueva sin la intervención de los administradores. Para ello, se considerará el número de plazas disponibles en el grupo de asignatura asociado a la máquina virtual.
- desplegar las modificaciones de una configuración sin que tengan que intervenir los administradores.
- borrar una de sus configuraciones de todo el sistema.

Por otro lado, los administradores pueden realizar las mismas tareas que los profesores. Además, pueden

- realizar las altas, bajas, arranques y apagados de los servidores que alojarán las máquinas virtuales.
- apagar todos los servidores del *cluster* de forma ordenada.
- realizar las altas y bajas de usuarios y grupos de asignaturas.
- fijar los privilegios de los distintos usuarios.
- desplegar manualmente una configuración en cierto servidor de máquinas virtuales.
- borrar manualmente una configuración de cierto servidor de máquinas virtuales.
- modificar la configuración básica de los servidores de máquinas virtuales y su uso durante el balanceado de carga.

Finalmente, todos los usuarios interactuarán con el sistema *CygnusCloud* utilizando una aplicación web que, además, realizará las labores de control de acceso para cada tipo de usuario y gestión de grupos de asignaturas.

6.2.2. Errores tratados

Para que *CygnusCloud* pueda utilizarse en una prueba piloto o incluso en producción, es necesario que la infraestructura detecte y trate un gran número de errores, que enumeraremos en esta sección. Por claridad, agruparemos estos errores en función del subsistema en el que se detecten.

En primer lugar, en el repositorio de imágenes se detectarán y tratarán errores cuando

- el servidor de *cluster* intenta borrar del repositorio un fichero comprimido que no existe.
- un servidor de máquinas virtuales intenta descargar o transferir al repositorio un fichero comprimido que no está asociado a ninguna configuración registrada.

- un servidor de máquinas virtuales intenta descargar un fichero comprimido que ha sido concedido en exclusividad a otro servidor de máquinas virtuales.
- un servidor de máquinas virtuales intenta sobrescribir un fichero comprimido que no está siendo modificado en ningún servidor de máquinas virtuales.

Por otra parte, en el servidor de máquinas virtuales se detectarán y tratarán errores cuando

- falla el hipervisor KVM.
- el servidor de *cluster* ordena el arranque de una máquina virtual que no está registrada.
- el servidor de *cluster* ordena la edición de una configuración que se está modificando en otro servidor de máquinas virtuales.
- falla la compresión o la descompresión de los ficheros `.zip`.
- no se puede establecer la conexión con el repositorio de imágenes para subir o descargar de él un fichero `.zip`. Para evitar que se pierdan los cambios en las configuraciones de las máquinas virtuales, el servidor de máquinas virtuales intentará restablecer la conexión en varias ocasiones antes de generar el error.
- falla la descarga o la subida de un fichero `.zip` al repositorio de imágenes. Para evitar que se pierdan los cambios en las configuraciones de las máquinas virtuales, el servidor de máquinas virtuales intentará volver a realizar la operación en varias ocasiones antes de generar el error.

Además, si el servidor de máquinas virtuales se apaga de forma brusca, en su siguiente arranque

- se detectarán las máquinas virtuales que siguen activas,
- se borrarán del disco los ficheros `.zip` parcialmente generados,
- se reiniciarán las transferencias con el repositorio de imágenes (para no perder los cambios que han hecho los usuarios en las configuraciones de las máquinas virtuales), y
- se liberarán todos los recursos asignados a las máquinas virtuales que se han apagado.

Por otro lado, en el servidor de *cluster* se detectarán y tratarán errores cuando

- un usuario intenta actualizar todas las copias de una imagen o configuración que no se ha modificado.
- un usuario ordena el despliegue automático de una imagen a la que afecta otra operación de despliegue anterior.
- a la hora de realizar el despliegue automático de la imagen, los servidores de máquinas virtuales no pueden albergar el número máximo de máquinas virtuales activas solicitado.
- un usuario intenta borrar una imagen que se está editando.
- un usuario intenta borrar una imagen que está afectada por una operación de borrado anterior.
- un usuario intenta arrancar una máquina virtual que no está registrada.
- un usuario intenta crear una imagen a partir de otra imagen que no existe.
- un usuario intenta editar una imagen que no existe.
- un usuario intenta borrar una imagen que no existe.
- para procesar la petición es necesario interactuar con el repositorio de imágenes, y la conexión con dicha máquina se ha perdido.

- el repositorio de imágenes no tiene suficiente espacio en disco como para crear una nueva imagen.
- se intenta apagar, parar o dar de baja un servidor de máquinas virtuales que no está registrado.
- se intenta enviar una petición a un servidor de máquinas virtuales que está arrancando.
- se intenta desplegar una imagen de disco en un servidor de máquinas virtuales que ya la almacena.
- se intenta borrar una imagen de disco de un servidor de máquinas virtuales que no la almacena.
- un servidor de máquinas virtuales no dispone de suficiente espacio en disco como para almacenar la imagen a desplegar.
- no se puede establecer la conexión con el servidor de máquinas virtuales al que afecta la petición.
- se ha perdido la conexión con el servidor de máquinas virtuales al que afecta la petición.
- un usuario intenta apagar o reiniciar una máquina virtual que no existe.
- se intenta modificar la configuración de un servidor de máquinas virtuales que ya está arrancado.
- el nuevo nombre o la nueva dirección IP asignados a un servidor de máquinas virtuales ya están en uso.
- el despliegue automático de una imagen falla en algunos servidores de máquinas virtuales.
- el borrado de una imagen falla en algunos servidores de máquinas virtuales.
- se excede el tiempo máximo de arranque de una máquina virtual.
- no hay ningún servidor de máquinas virtuales que pueda albergar la máquina virtual que se pretende arrancar.
- no hay servidores de máquinas virtuales en los que se pueda editar la imagen seleccionada.
- no hay servidores de máquinas virtuales en los que pueda desplegarse una imagen. Este error estará asociado a las operaciones de despliegue automático.
- todos los servidores de máquinas virtuales que se usan para crear o editar imágenes no disponen de recursos suficientes como para editar una imagen más.

Finalmente, el servidor web detectará y tratará las caídas del servidor de *cluster*. En estos casos,

- intentará restablecer la conexión con el mismo, y
- cuando se exceda el tiempo máximo de procesamiento de una petición, generará el error correspondiente

6.3. Problemas encontrados durante el desarrollo

Durante el desarrollo de *CygnusCloud*, hemos tenido que resolver dos situaciones problemáticas: la necesidad de cambiar de hipervisor y el uso de una librería de red pésimamente documentada.

6.3.1. Cambio de hipervisor

En la fase inicial del proyecto, evaluamos los hipervisores Xen y KVM. Puesto que ambos cuentan con el mismo rendimiento y, en principio, soportan las mismas funciones, decidimos utilizar Xen por su mayor soporte comercial.

Así, estuvimos familiarizándonos con este hipervisor durante una semana. En ese periodo de tiempo, lo instalamos y lo utilizamos para crear distintas máquinas virtuales.

Tras adquirir suficiente experiencia, decidimos empezar a utilizar imágenes *copy-on-write*. Esto marcó el inicio de todos nuestros problemas con Xen.

En primer lugar, encontramos un *bug* en el parser de los ficheros de definición: los dos elementos XML que se utilizan para indicar la ruta de la imagen de disco no pueden estar separados por espacios, tabuladores ni líneas en blanco. Para averiguar la causa de este *bug*, tuvimos que invertir cerca de una semana, en la que examinamos el código que se utilizaba para interactuar con Xen.

Por suerte para nosotros, ese código está escrito en *Python*, por lo que es pudimos modificarlo rápidamente para hacer pruebas.

Una vez resuelto este inconveniente descubrimos, también perdiendo cerca de una semana, que la versión *open-source* de Xen no soporta todas las características que se indican en su documentación oficial. Aunque esta afirma que Xen soporta imágenes de disco RAW, `qcow`, `qcow2` y `vmdk`, en la práctica sólo es posible utilizar imágenes RAW y `qcow` en modo no *copy-on-write*. Asimismo, en ninguno de los escasos foros y listas de correo que contienen información sobre Xen se mencionaba este aspecto.

En un último intento, decidimos compilar manualmente la versión más reciente de Xen, la 4.2. Tras hacerlo y comprobar que

- estos problemas seguían estando presentes,
- la documentación de la nueva versión era prácticamente inexistente, y
- hasta la publicación de la nueva versión de Xen, que tendría lugar a mediados de 2013, los integrantes del proyecto Xen no tenían intención de resolver ningún *bug*,

decidimos utilizar KVM. Esta ha sido una de las mejores decisiones de diseño que hemos tomado: tras un fin de semana, pudimos iniciar, con un mes de retraso, el desarrollo de los servidores de máquinas virtuales.

6.3.2. La librería de red *twisted*

Cuando comenzamos a trabajar en el proyecto, decidimos utilizar una librería de red escrita en *Python* y, de ser posible, muy utilizada. En caso de tener problemas, esto último nos permitiría obtener mucha información.

Así, nos decantamos por utilizar la librería de red *twisted*. Aunque encontramos bastantes tutoriales para aprender a utilizar esta librería, ninguno de ellos profundizaba lo suficiente como para cubrir nuestras necesidades.

Por ello, decidimos tomar los tutoriales como base y, a partir de la documentación oficial de *twisted*, extenderlos para implementar el módulo de comunicaciones.

Durante el desarrollo de este módulo observamos que, aunque utilizásemos la última versión de *twisted* y la versión más actualizada de su documentación, muchos de los ejemplos incluidos en la misma no funcionaban.

Para confirmar que la documentación de *twisted* se contradice a sí misma, decidimos utilizar la función `dir()` de *Python*. Al evaluarse, esta función devuelve una lista con todos los atributos de un objeto.

Aunque existen muchos foros en los que se resuelven dudas sobre el uso de la librería *twisted*, en la inmensa mayoría de ellos no se indica la versión de *twisted* que se está utilizando, Por tanto, para utilizar la librería *twisted* es necesario recurrir al método de prueba y error en bastantes ocasiones.

Cuando detectamos este inconveniente, habíamos invertido un esfuerzo considerable en comprender el funcionamiento de *twisted* y en implementar el módulo de comunicaciones. Por ello, decidimos seguir utilizando esta librería de red.

6.4. Repercusiones

El esfuerzo que hemos invertido en *CygnusCloud* se ha visto recompensado de varias formas.

En primer lugar, un artículo en el que se mostraban las ideas básicas de *CygnusCloud* y del proyecto *SmartCloud* fue portada de la prestigiosa publicación *HPC In The Cloud* durante más de un mes. Dicho artículo puede consultarse en

http://www.hpcinthecloud.com/hpccloud/2013-01-04/student_projects_highlight_cloud_s_-potential.html

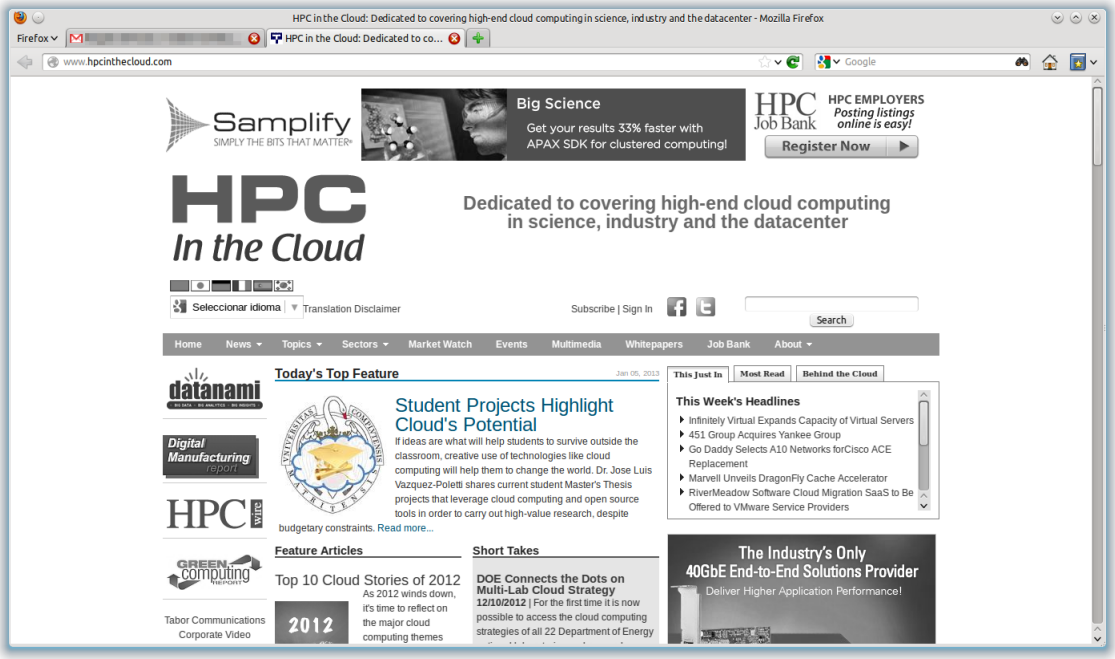

FIGURA 6.1: Portada de la publicación HPC In The Cloud en enero de 2013

Tras la publicación de ese artículo, fuimos entrevistados junto con los miembros de los proyectos *SmartCloud* y *Horadrim* en el programa Hoy por hoy Madrid de la Cadena Ser. En dicho espacio, hablamos de las alternativas que ofrece *CygnusCloud* y el *Cloud Computing* en general como solución a la actual crisis económica. El audio de esta entrevista está disponible en

http://tinyurl.com/ox428wq

También, a lo largo del curso 1012/13, la web de la Facultad de Informática de la Universidad Complutense de Madrid ha publicado un par de artículos sobre nuestro proyecto. Ambos artículos se encuentran disponibles en

http://www.fdi.ucm.es/Documento.asp?cod=1006

http://www.fdi.ucm.es/Documento.asp?cod=1056

Por si fuera poco, el periódico de la Universidad Complutense, Tribuna Complutense, publicó un artículo en el que se mencionan los objetivos principales de *CygnusCloud*. Puede leerse en

http://cygnusclouducm.files.wordpress.com/2013/01/tribunacomplutense.pdf

Por otra parte, *CygnusCloud* recibió el premio al mejor proyecto comunitario en la final nacional de la séptima edición del Concurso Universitario de Software Libre, que se celebró en Granada el día 24 de mayo de 2013. La lista completa de premiados en ese evento está disponible en

http://www.concursosoftwarelibre.org/1213/premiados-vii-cusl

FIGURA 6.2: Foto de familia de la final nacional de la 7ª edición Concurso Universitario de Software Libre. Los miembros de *CygnusCloud* aparecen en la segunda fila, a la izquierda. *Fuente: Concurso Universitario de Software Libre*

Durante ese evento, profesores de la Universidad de Granada nos comunicaron que estudiarán la posible implantación del sistema *CygnusCloud*.

Además, a lo largo del proceso de desarrollo hemos mantenido un *blog*, que está disponible en

http://cygnusclouducm.wordpress.com/

A través de él, hemos resuelto las dudas de varios usuarios. Asimismo, algunas entradas del *blog*, como la comparativa entre Xen y KVM, *aparecen entre los primeros resultados al hacer consultas sobre sus temas en Google.*

Finalmente, a fecha de 13 de Junio de 2013, el *blog* de *CygnusCloud* cuenta con un total de 4657 visitas. En la figura 6.3 se recoge la distribución geográfica de la mayor parte de este tráfico.

Country	Views
Spain	2,748
Mexico	354
Argentina	347
Colombia	248
Chile	190
Peru	166
Venezuela	123
Ecuador	94
Costa Rica	40
United States	39
El Salvador	33
Guatemala	27
Uruguay	24
Paraguay	21
Bolivia	21
Cuba	17
Nicaragua	17
Dominican Republic	16
Panama	16
Honduras	13
Germany	11
United Kingdom	9
Czech Republic	8
Netherlands	7
Puerto Rico	4
France	4
Japan	3

FIGURA 6.3: Distribución geográfica de las visitas al *blog* de *CygnusCloud*

Capítulo 7

Trabajo futuro

Durante el presente curso académico, hemos tenido que compaginar el desarrollo de *CygnusCloud* con la realización de prácticas y la preparación de exámenes de otras asignaturas.

Por otra parte, cuando decidimos presentar este proyecto a la séptima edición del Concurso Universitario de Software Libre, tuvimos que cumplir con las bases del mismo. Esto ha supuesto

- la creación y actualización del *blog*, en el que hemos reflejado las decisiones de diseño más relevantes que hemos tomado, y
- la creación de la versión *release* que ha utilizado el jurado del concurso para evaluar nuestro proyecto.

Asimismo, como hemos visto en el capítulo anterior, durante el proceso de desarrollo nos hemos tenido que enfrentar a un cambio de hipervisor, lo que supuso, literalmente, perder un mes de trabajo.

Además, cuando iniciamos el desarrollo de *CygnusCloud* nos propusimos que la versión final fuese lo suficientemente robusta y estable como para ser utilizada en una prueba piloto o incluso en producción.

Por todos estos motivos, hemos tenido que dejar en el tintero muchas ideas que han ido surgiendo en el proceso de desarrollo.

En este capítulo, sugeriremos un conjunto de modificaciones que pueden incorporarse a la versión final de *CygnusCloud*.

7.1. Soporte *multi-cluster*

Como dijimos al presentar la arquitectura del sistema, el proceso *endpoint*, que comunica la aplicación web y la infraestructura a través de una base de datos, sólo se conectará a un único servidor de *cluster*.

A medida que aumenta el número de usuarios, también aumentará el número de máquinas virtuales que estarán activas a la vez. Para albergarlas, habrá que asignar más servidores de máquinas virtuales al *cluster*.

Por tanto, a medida que aumenta el número de usuarios también aumentará el número de servidores de máquinas virtuales a los que el servidor de *cluster* tendrá que atender simultáneamente. Como los recursos de los que dispone el servidor de *cluster* son limitados, llegará un momento en el que no será posible añadir más servidores de máquinas virtuales al *cluster*.

Para superar esta limitación, es preciso añadir más *clusters* a la infraestructura de *CygnusCloud*, y modificar el proceso *endpoint* del servidor web para que, en tiempo de ejecución, pueda interactuar con varios servidores de *cluster*.

A la hora de distribuir las peticiones entre los servidores de *cluster*, el proceso *endpoint* deberá utilizar un algoritmo de balanceado de carga.

Es importante notar que, en este caso, sólo es necesario realizar modificaciones en el proceso *endpoint*. El resto de subsistemas de *CygnusCloud* podrán utilizarse directamente tras realizar esta optimización.

7.2. Uso de varios servidores web

Los recursos del servidor web también están limitados. Por ello, a medida que el número de usuarios de *CygnusCloud* crece, llegará un momento en el que el servidor *web* no podrá atenderlos a todos.

Para superar esta limitación, es necesario utilizar varios servidores web. Puesto que la infraestructura de *CygnusCloud* utiliza identificadores únicos de petición muy largos cuyo contenido puede ser arbitrario, para utilizar varios servidores *web* sólo será necesario modificar el proceso *endpoint* y modificar ligeramente el servidor de *cluster*.

En el proceso *endpoint* hay que introducir dos cambios muy sencillos:

- debemos modificar el algoritmo que genera los identificadores únicos de las peticiones, de forma que podamos garantizar que dos servidores web distintos nunca generen peticiones con el mismo identificador único.
- para ahorrar ancho de banda, sólo uno de los procesos *endpoint* enviará los paquetes de solicitud de estado al servidor de *cluster*. El resto, sólo procesará los datos de estado enviados por el servidor de *cluster*.

Por otra parte, las modificaciones que hay que introducir en el servidor de *cluster* son triviales:

- los paquetes que contienen información de estado se enviarán a los servidores web, como hasta ahora, en modo *multicast*. Así, todos ellos los recibirán.
- el resto de paquetes se enviarán en modo *unicast* a los servidores de máquinas virtuales. Así, sólo el servidor de máquinas virtuales que emitió la petición recibirá la respuesta.

  Esta última modificación nos permite ahorrar mucho ancho de banda. Por ejemplo, cuando la infraestructura genera un error sólo el servidor web que envió la petición, y no todos, recibirán el paquete correspondiente.

7.3. Optimizaciones en el servidor de *cluster*

El servidor de *cluster* procesa todas las peticiones que recibe de forma secuencial. Si utilizamos como servidor de *cluster* una máquina relativamente moderna, esta tendrá una CPU *dual-core* o *quad-core* que incluso puede soportar *multithreading* simultáneo.

Aunque en el demonio del servidor de *cluster* ya existen varios hilos, podemos aprovechar aún más la potencia del *hardware* si procesamos las peticiones de los usuarios en paralelo.

Para ello, es necesario

- utilizar varios hilos de procesamiento de peticiones, y
- garantizar que todas las actualizaciones de la base de datos del servidor de *cluster* sean atómicas.

Es importante notar que, para aplicar esta optimización, no es necesario modificar el proceso *endpoint* ni el resto de subsistemas de *CygnusCloud*.

7.4. Cambio de la arquitectura del servidor web

Como sabemos, la aplicación web interactúa con la infraestructura de *CygnusCloud* a través de dos bases de datos. Por ello, resulta muy sencillo crear otra aplicación web totalmente diferente e interactuar con *CygnusCloud* a través de ella.

Esta nueva aplicación web no tiene por qué utilizar el *framework* `web2py` ni estar escrita en *Python*: puede estar escrita en cualquier lenguaje de programación para el que exista un conector de MySQL. Al implementarla, sólo es necesario serializar las peticiones y deserializar sus resultados adecuadamente.

Por otra parte, también es posible prescindir del proceso *endpoint*: siempre que se le envíen paquetes de los tipos adecuados, el servidor de *cluster* procesará sin problemas las peticiones correspondientes.

Todas estas características hacen que el sistema *CygnusCloud* pueda integrarse fácilmente en muchos entornos.

7.5. Uso de varios repositorios de imágenes en cada *cluster*

Cada *cluster* tiene asociado un repositorio de imágenes. En esta máquina, se almacenarán todas las configuraciones o imágenes que pueden utilizar las máquinas virtuales del *cluster*.

Puesto que el repositorio de imágenes interviene en todas las operaciones de creación, edición y despliegue de imágenes, se convertirá en un cuello de botella.

Si

- el *cluster* contiene un elevado número de servidores de máquinas virtuales, o si
- las operaciones de creación, edición y despliegue de imágenes son muy frecuentes,

podremos mejorar su funcionamiento utilizando varios repositorios de imágenes. Los cambios que hay que introducir en este caso son los siguientes:

- hay que modificar la forma en que se generan los identificadores de imagen, garantizando que dos repositorios distintos nunca puedan generar el mismo identificador.
- hay que modificar el servidor de *cluster* para que
  - se conecte a múltiples repositorios de imágenes,
  - recopile el estado de todos ellos, y
  - realice el balanceado de carga entre estas máquinas.

Finalmente, puesto que los servidores de máquinas virtuales reciben los datos de conexión al repositorio de imágenes en todas las peticiones de creación y edición de imágenes, no es preciso incluir ninguna modificación en este subsistema de *CygnusCloud*.

7.6. Cancelar la creación o edición de una imagen

Como sabemos, en los procesos de creación y edición de imágenes de disco se realizan dos transferencias de ficheros entre un servidor de máquinas virtuales y el repositorio de imágenes.

Puesto que los ficheros que intercambian estas máquinas contienen imágenes de disco, estas transferencias requieren mucho tiempo y ancho de banda.

Para no desperdiciar el ancho de banda disponible, resulta muy interesante que los usuarios puedan cancelar la ejecución de las operaciones de creación y edición de imágenes. Para ello, basta con introducir los siguientes cambios:

- cuando reciban la petición correspondiente, los servidores de máquinas virtuales deben abortar las transferencias pendientes con el repositorio de imágenes, liberar la imagen y borrar los ficheros correspondientes de su disco duro.

- el servidor de *cluster* procesará un nuevo tipo de petición. Cuando reciba el paquete correspondiente, actualizará su base de datos y avisará al servidor de máquinas virtuales en el que se esté editando la imagen.

- el proceso *endpoint* será capaz de procesar un nuevo tipo de petición. Esencialmente, debe deserializarla, enviar el paquete correspondiente al servidor de *cluster* y procesar los nuevos tipos de error generados por el servidor de *cluster*.

- finalmente, la aplicación web debe ser capaz de realizar una nueva actualización de la base de datos de comandos. Asimismo, también será necesario modificar la interfaz de usuario para incluir en ella la nueva petición.

7.7. Uso de otro formato de fichero comprimido

Para ahorrar ancho de banda, el repositorio de imágenes y los servidores de máquinas virtuales intercambian las imágenes de disco de las distintas configuraciones en formato comprimido.

Aunque hemos decidido utilizar el formato `zip` para crear estos ficheros, resulta sencillo utilizar otro formato distinto.

Para ello, sólo es necesario modificar la clase que crea y extrae los ficheros comprimidos en el servidor de máquinas virtuales. Puesto que el repositorio de imágenes nunca extrae los ficheros comprimidos, no es necesario realizar ninguna modificación en él.

7.8. Cambio del algoritmo de balanceado de carga

Como hemos visto en el capítulo de arquitectura, el servidor de *cluster* utiliza la interfaz `LoadBalancer` para interactuar con las instancias de las clases asociadas a los algoritmos de balanceado de carga.

Por ello, resulta muy sencillo implementar nuevos algoritmos de balanceado de carga: bastará con crear nuevas clases que implementen esa interfaz.

Como vimos en su momento, los algoritmos de balanceado de carga podrán trabajar con esta información:

- número de CPUs virtuales, tamaño de la memoria RAM, tamaño de la imagen del sistema operativo y tamaño de la imagen con los datos temporales de la máquina virtual a crear.

- número de instancias, número de CPUs virtuales, número de CPUs físicas, memoria RAM ocupada, memoria RAM total, espacio de almacenamiento ocupado, espacio de almacenamiento total, espacio de almacenamiento temporal ocupado y espacio de almacenamiento temporal total en todos los servidores de máquinas virtuales arrancados.

- modo de funcionamiento (normal o reservado para crear y editar imágenes) de todos los servidores de máquinas virtuales arrancados.

7.9. Uso de dos límites temporales en el proceso *endpoint*

Cuando se produce una caída del servidor de *cluster*, el proceso *endpoint* generará errores para las distintas peticiones mediante un mecanismo basado en *timeouts*.

Ahora bien, en la implementación actual se utiliza un único límite temporal. Por ello, si no se ajusta de forma suficientemente cuidadosa, pueden generarse errores por *timeout* para las peticiones de alta latencia (como la creación y el despliegue de imágenes).

Por ello, resulta interesante modificar el proceso *endpoint* para que, a la hora de generar errores por *timeout*, utilice dos límites temporales: uno para las operaciones de alta latencia y otro para el resto de operaciones.

7.10. Registro de imágenes base desde la aplicación web

Tras instalar los módulos de *CygnusCloud* en las máquinas del *cluster*, no existe ninguna configuración registrada. Para que los usuarios puedan empezar a utilizar la infraestructura, los administradores tendrán que crear y registrar manualmente una o más imágenes base en el *cluster*. Para ello, es necesario que

- los administradores creen las imágenes de disco fuera del sistema *CygnusCloud*, que
- generen los ficheros comprimidos que hay que transferir al repositorio de imágenes, que
- transfieran esos ficheros al repositorio de imágenes (mediante una utilidad de conexión al repositorio), y que
- registren las nuevas imágenes base en la base de datos del servidor de *cluster* y en la base de datos del *endpoint*.

Puesto que las imágenes base se transfieren mediante el protocolo FTP, es posible integrar la mayor parte este proceso en la aplicación web. Para ello, es necesario

- modificar el proceso *endpoint* para que interactúe con el repositorio de imágenes y realice las actualizaciones correspondientes en la base de datos del *endpoint*,
- modificar la aplicación web para que los administradores puedan transferir ficheros comprimidos al servidor web, y
- modificar el servidor de *cluster* añadiendo una petición que registra identificadores de imágenes de disco.

De esta manera, los administradores sólo tendrán que ocuparse de la creación manual de las imágenes de disco, lo que simplifica mucho su labor y, sobre todo, evita la aparición de errores.

7.11. Detección de errores por *timeout* en el servidor de *cluster*

Como sabemos, el servidor de *cluster* monitoriza todos los comandos de arranque de máquinas virtuales. De esta manera, cuando el tiempo que transcurrido desde que se envió la petición al servidor de máquinas virtuales alcanza cierto umbral, el servidor de *cluster* asume que esta ha fallado e informará del error.

Para tratar las desconexiones de los servidores de máquinas virtuales de forma más adecuada, podemos ampliar este mecanismo, de modo que puedan generarse errores por *timeout* para todas las peticiones recibidas (y no sólo para las peticiones de arranque de máquinas virtuales).

Capítulo 8

Manual de usuario

8.1. Requisitos mínimos

Cada servidor de la infraestructura de *CygnusCloud* desempeña uno de estos papeles:

- **servidor de máquinas virtuales**. Estos equipos alojan las máquinas virtuales activas de los usuarios. Asimismo, estas máquinas también se utilizan para crear y editar las configuraciones de las máquinas virtuales.
- **servidor de *cluster***. Estas máquinas realizan el balanceado de carga entre varios servidores de máquinas virtuales. También se ocupan de la gestión de estas máquinas.
- **repositorio de imágenes**. Estas máquinas almacenan las configuraciones de todas las máquinas virtuales que pueden utilizarse en la infraestructura.
- **servidor web**. Estas máquinas proporcionan una interfaz web que permite a los usuarios enviar peticiones a los servidores de *cluster*.

En esta sección, empezaremos mostrando los requisitos mínimos que debe cumplir el *hardware* de cada una de estas máquinas, para posteriormente mostrar los requisitos *software*.

8.1.1. Requisitos *hardware*

Las máquinas que se utilicen como servidores de *cluster* o servidores web deben cumplir los siguientes requisitos mínimos:

- CPU Intel® Pentium 4® a 2,4 GHz o superior
- un mínimo de 512 *megabytes* de memoria RAM. Se recomienda 1 GB para un funcionamiento óptimo.
- un *gigabyte* de espacio en disco
- tarjeta de red *fast ethernet*.

Por otra parte, las máquinas que se utilicen como repositorios de imágenes tienen unos requisitos mínimos muy similares:

- CPU Intel® Pentium 4® a 2,4 GHz o superior
- un mínimo de 512 *megabytes* de memoria RAM. Se recomienda 1 *gigabyte* para un funcionamiento óptimo.
- un mínimo de 20 *gigabytes* de espacio en disco.
- tarjeta de red *fast ethernet*.

Por otro lado, las máquinas que se utilicen como servidores de máquinas virtuales deben cumplir los siguientes requisitos mínimos:

- CPU Intel® Core 2 Duo® E4400 o superior
- un mínimo de 2 *gigabytes* de memoria RAM. Se recomiendan 4 GB para un funcionamiento óptimo.
- un mínimo de 30 *gigabytes* de espacio en disco si sólo se desea alojar máquinas virtuales Linux. Si también se pretende alojar máquinas virtuales *Windows*, se requiere un mínimo de 60 *gigabytes* de espacio en disco.
- tarjeta de red *fast ethernet*.

Finalmente, todas las máquinas de la infraestructura deben estar conectadas a una misma red, que deberá soportar, como mínimo, una tasa de transferencia de 100 *megabits* por segundo.

8.1.2. Requisitos *software*

Todas las máquinas de la infraestructura deben tener instalado el sistema operativo *Linux*.

En principio, los demonios de *CygnusCloud* deberían funcionar de forma correcta en cualquier distribución *Linux* publicada tras el mes de abril del año 2012. No obstante, por falta de tiempo, sólo hemos podido hacer pruebas con las distribuciones Ubuntu 12.04 LTS, Ubuntu 13.04, Debian 7, openSUSE 12.2 y openSUSE 12.3.

En todas las máquinas de la infraestructura deberán estar instalados, además del sistema operativo,

- una versión de la rama 2.7 del intérprete de *Python*, igual o superior a la 2.7.3.
- la librería de red *twisted*, versión 12.1.0 o superior.
- el gestor de bases de datos MariaDB, versión 5.5.31 o superior.
- las utilidades `mv`, `cp` y `rm`.
- el conector *Python* de MySQL. Este puede descargarse desde

  `http://dev.mysql.com/downloads/connector/python/`
- el conector MySQL `mysqldb`.
- la orden `openssl`.

Importante: a causa de las limitaciones impuestas por `libvirt` y por la librería *twisted*, *no* es posible utilizar un intérprete de *Python* de la rama 3.x.

Por otra parte, en los repositorios de imágenes también deberán estar instalados

- el servidor FTP `pytfpdlib`. Este puede descargarse desde

  `http://code.google.com/p/pyftpdlib/`
- la utilidad `ethtool`.

Finalmente, en los servidores de máquinas virtuales también debe estar instalado el siguiente *software* adicional:

- un *kernel* de la versión 3.2.0 o superior.
- el hipervisor de KVM correspondiente a la versión del *kernel* que se encuentra instalada.
- el sistema de virtualización QEMU, versión 1.4 o superior
- la librería `libvirt`, versión 1.0.6 o superior
- las utilidades `zip` y `unzip`.

8.2. Instrucciones de instalación y configuración en Ubuntu 12.04

La última versión de todos los subsistemas y utilidades auxiliares de `CygnusCloud` puede descargarse desde

http://cygnusclouducm.wordpress.com/descargas/

Al exponer el proceso de instalación de los distintos subsistemas de *CygnusCloud*, asumiremos que el fichero .tar.gz que contiene los módulos de dicho subsistema ya se ha descargado en la máquina que se pretende utilizar.

Por ejemplo, en el proceso de instalación del repositorio de imágenes, omitiremos el paso inicial, en el que se descarga el fichero comprimido `ImageRepository.tar.gz` desde la página web http://cygnusclouducm.wordpress.com/descargas/.

8.2.1. Pasos comunes

Con independencia del papel que desempeñe la máquina en la infraestructura, siempre es necesario

1. instalar el gestor de bases de datos MariaDB. Para ello, ejecutamos las órdenes

```
sudo apt-get install python-software-properties
sudo apt-key adv --recv-keys
    --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
sudo add-apt-repository
    'deb http://mariadb.cu.be//repo/5.5/ubuntu precise main'
sudo apt-get update sudo apt-get install mariadb-server
```

2. instalar el conector oficial de MySQL. Para ello,

   a) descargamos el fichero `.tar` desde

   http://dev.mysql.com/downloads/connector/python/

   Para que aparezca el enlace de descarga, debemos seleccionar Source Code en el *combobox* que aparece en la página.

   b) extraemos el fichero `.tar` mediante la orden

```
tar xvzf mysql-connector-python-1.0.10.tar.gz -C <ruta del directorio de
                                    extracción>
```

   c) navegamos hasta el directorio en el que hemos extraído el fichero `.tar`. Acto seguido, ejecutamos los comandos

```
python setup.py build
sudo python setup.py install
```

3. instalar la librería de red *twisted*. y el conector MySQL `MysqlDB`. Para ello, ejecutamos la orden

```
sudo apt-get install python-twisted python-mysqldb
```

8.2.2. Repositorio de imágenes

8.2.2.1. Instalación y configuración del software adicional requerido

Una vez instalados los componentes comunes, en el repositorio de imágenes también es necesario instalar el servidor FTP `pyftpdlib` y la orden `ethtool`. Para ello, seguimos los siguientes pasos:

1. descargamos el fichero `.tar` desde

   http://pyftpdlib.googlecode.com/files/pyftpdlib-1.0.1.tar.gz

2. extraemos dicho fichero. Para ello, ejecutamos la orden

   ```
   tar xvzf pyftpdlib-1.0.1.tar.gz -C <ruta del directorio de extracción>
   ```

3. navegamos hasta el directorio en el que hemos extraído el contenido del fichero `.tar`. Acto seguido, ejecutamos los comandos

   ```
   python setup.py build
   sudo python setup.py install
   ```

4. ejecutamos el comando

   ```
   sudo apt-get install ethtool
   ```

8.2.2.2. Instalación del módulo de *CygnusCloud*

Los pasos a seguir son los siguientes:

1. descomprimimos el fichero `ImageRepository.tar.gz` en el directorio de instalación. Para ello, ejecutamos la orden

   ```
   tar xvzf ImageRepository.tar.gz -C <ruta del directorio de instalación>
   ```

2. tomamos posesión de todos los ficheros extraídos. Para ello, ejecutamos la orden

   ```
   sudo chown -R <nuestro usuario>:<nuestro usuario> <ruta del directorio de
                                   instalación>
   ```

3. ejecutamos el *script* `build.py`, ubicado en la raíz del directorio de instalación. Para ello, utilizamos el comando

   ```
   python build.py
   ```

   o el comando

   ```
   sh build.py
   ```

   El *script* comprobará que se satisfacen todas las dependencias e informará de los errores que detecte.

4. si se satisfacen las dependencias, el *script* generará el fichero `imageRepository.sh`. Este fichero es el *script* de arranque del demonio del repositorio de imágenes.

8.2.2.3. Configuración del módulo de *CygnusCloud*

Si intentamos arrancar el demonio del repositorio de imágenes sin haber inicializado previamente su fichero de configuración, se generará un mensaje de error. El fichero de configuración de este demonio, `ImageRepository_settings.conf`, se encuentra en la raíz del directorio de instalación.

El formato de todos los ficheros de configuración es el siguiente:

- las líneas que empiezan por almohadilla (`#`) o por punto y coma (`;`) son comentarios. Los comentarios de línea también están soportados.
- las cadenas de caracteres rodeadas por corchetes (`[]`) son encabezados de sección. **No** deben modificarse jamás.
- el resto de líneas del fichero de configuración son de la forma `parámetro = valor`. Si no aparece ningún valor a la derecha del símbolo `=`, el valor asociado al parámetro será la cadena vacía.

El contenido inicial del fichero de configuración del repositorio de imágenes aparece en la figura 8.1.

```
# Comment or delete the following line after initializing this file
[Uninitialized file]
[Database configuration]
mysqlRootsPassword =
dbUser = cygnuscloud
dbUserPassword = cygnuscloud
[Network configuration]
useSSL = False
certificatePath = /home/luis/Infraestructura/Certificates
commandsPort = 3000
[FTP Server configuration]
FTPListenningInterface = eth0
FTPPort = 2121
maxConnections = 10
maxConnectionsPerIP = 1
uploadBandwidthRatio = 0.9
downloadBandwidthRatio = 0.9
FTPRootDirectory = /home/luis/Infraestructura/ImageRepository
FTPPasswordLength = 20
FTPUserName = cygnuscloud
```

FIGURA 8.1: Contenido inicial del fichero `ImageRepository_settings.conf`

En primer lugar, la sección `Database configuration` especifica la información que se utilizará para configurar la base de datos del repositorio de imágenes: la contraseña del usuario *root* y el nombre y la contraseña del usuario que empleará el demonio.

Por otra parte, la sección `Network configuration` especifica la configuración de la conexión de control. En ella, se indicará si se desea cifrar el tráfico de esta conexión, la ruta de los certificados `server.crt` y `server.key` (sólo se utilizará cuando se cifra el tráfico) y el puerto asociado a la conexión de control.

Asimismo, la sección `FTP Server configuration` especifica la configuración del servidor FTP que reside en el repositorio de imágenes. En ella, se indicará

- la interfaz de red por la que circulará el tráfico FTP.

- el puerto del servidor FTP.
- el número máximo de conexiones activas en total y el número máximo de conexiones activas de cada dirección IP.
- la fracción del ancho de banda de bajada y del ancho de banda de subida que podrá ser utilizada por el tráfico FTP.
- la ruta del directorio raíz del servidor FTP.
- el nombre de usuario y la longitud de la contraseña a utilizar. Las contraseñas se generan aleatoriamente en cada arranque.

Finalmente, tras inicializar el contenido de este fichero, debemos borrar o comentar la sección `Uninitialized file`.

8.2.3. Servidores de máquinas virtuales

8.2.3.1. Instalación y configuración de KVM

Una vez instalados los paquetes comunes, en los servidores de máquinas virtuales debemos instalar el hipervisor KVM, la librería `libvirt` y el sistema de virtualización QEMU. Para ello, ejecutamos la orden

```
sudo apt-get install qemu-kvm libvirt-bin python-libvirt ubuntu-vm-builder
                                bridge-utils
```

Tras instalar estos paquetes, es necesario añadir el usuario que vamos a utilizar a los grupos `libvirtd` y `kvm`. Para ello, ejecutamos las órdenes

```
sudo addgroup <nombre de nuestro usuario> libvirtd
```

y

```
sudo addgroup <nombre de nuestro usuario> kvm
```

Además, debemos configurar `libvirt` para que nos permita crear máquinas y redes virtuales sin necesidad de ser *root*. Para ello, descomentamos las siguientes líneas del fichero `/etc/libvirt/libvirtd.conf`:

- `unix_sock_group = "libvirtd"`
- `unix_sock_ro_perms = "0777"`
- `unix_sock_rw_perms = "0770"`
- `auth_unix_ro = "none"`
- `auth_unix_rw = "none"`

Finalmente, reiniciamos el demonio de `libvirt` mediante el comando

```
sudo service libvirtd restart
```

y reiniciamos nuestra sesión. A partir de este momento, podremos crear máquinas virtuales con nuestro usuario.

8.2.3.2. Instalación del módulo de *CygnusCloud*

Los pasos a seguir son los siguientes:

1. descomprimimos el fichero `VirtualMachineServer.tar.gz` en el directorio de instalación. Para ello, ejecutamos la orden

   ```
   tar xvzf VirtualMachineServer.tar.gz -C <ruta del directorio de instalación>
   ```

2. tomamos posesión de todos los ficheros extraídos. Para ello, ejecutamos la orden

   ```
   sudo chown -R <nuestro usuario>:<nuestro usuario> <ruta del directorio de
                                   instalación>
   ```

3. ejecutamos el *script* `build.py`, ubicado en la raíz del directorio de instalación. Para ello, utilizamos el comando

   ```
   python build.py
   ```

   o el comando

   ```
   sh build.py
   ```

   El *script* comprobará que se satisfacen todas las dependencias e informará de los errores que detecte.

4. si se satisfacen las dependencias, el *script* generará el fichero `virtualMachineServer.sh`. Este fichero es el *script* de arranque del demonio del servidor de máquinas virtuales.

8.2.3.3. Configuración del módulo de *CygnusCloud*

Si intentamos arrancar el demonio del servidor de máquinas virtuales sin haber inicializado previamente su fichero de configuración, se generará un mensaje de error. El fichero de configuración de este demonio, `VMServer_settings.conf`, se encuentra en la raíz del directorio de instalación. Su contenido inicial aparece en la figura 8.2.

Debemos recordar que, tras inicializar el contenido de este fichero, debemos borrar o comentar la sección `Uninitialized file`.

En primer lugar, la sección `Database configuration` especifica la información que se utilizará para configurar la base de datos del servidor de máquinas virtuales: la contraseña del usuario *root* y el nombre y la contraseña del usuario que empleará el demonio.

Por otra parte, la sección `Virtual Network Configuration` especifica la configuración de la red virtual que alberga el servidor de máquinas virtuales. En ella, se indica

- si la red virtual se debe crear como un usuario normal o como el usuario *root*,
- el nombre de la red virtual,
- la dirección IP y la máscara de red del encaminador predeterminado de dicha red, y
- el rango de direcciones IP que asignará el servidor DHCP.

Por otro lado, la sección `VNC server configuration` especifica

- la interfaz de red por la que circulará el tráfico VNC,
- la longitud de las contraseñas de los servidores VNC que residen en este servidor de máquinas virtuales,
- qué *software* utilizamos para crear los *websockets* (QEMU o `websockify`), y

- la ruta del proceso `websockify`. El valor de este último parámetro sólo se utiliza cuando los *websockets* se crean mediante `websockify`.

La sección `FTP Client Configuration` especifica el *timeout* y el número de veces que se reintentará realizar una transferencia fallida con el repositorio de imágenes.

Asimismo, la sección `Network configuration` especifica la configuración de la conexión de control. En ella, se indicará si se desea cifrar el tráfico de esta conexión, la ruta de los certificados `server.crt` y `server.key` (sólo se utilizará cuando se cifra el tráfico) y el puerto asociado a la conexión de control.

```
# Comment or delete the following line after initializing this file
[Uninitialized file]
[Database configuration]
mysqlRootsPassword =
databaseUserName = cygnuscloud
databasePassword = cygnuscloud
[Virtual Network Configuration]
createVirtualNetworkAsRoot = True
vnName = ccnet
gatewayIP = 192.168.77.1
netMask = 255.255.255.0
dhcpStartIP = 192.168.77.2
dhcpEndIP = 192.168.77.254
[VNC server configuration]
vncNetworkInterface = eth0
passwordLength = 46
useQEMUWebsockets = False
websockifyPath = /home/luis/Infraestructura/websockify
[FTP Client Configuration]
FTPTimeout = 100
MaxTransferAttempts = 5
[Network configuration]
useSSL = False
certificatePath = /home/luis/Infraestructura/Certificates
listenningPort = 15800
[Paths]
configFilePath = /home/luis/configuraciones/
sourceImagePath = /home/luis//imagenes/
executionImagePath = /home/luis/imagenes_en_ejecucion/
TransferDirectory = /home/luis/Transfers
```

FIGURA 8.2: Contenido inicial del fichero `VMServer_settings.conf`

Finalmente, la sección `Paths` especifica las rutas de los diversos directorios de trabajo del servidor de máquinas virtuales en los que se almacenan

- los ficheros de definición, los ficheros de imagen y los ficheros de imagen que utilizan las máquinas virtuales durante su ejecución, y
- los ficheros `.zip` que se intercambian con el repositorio de imágenes.

8.2.4. Servidores de cluster

8.2.4.1. Instalación del módulo de *CygnusCloud*

La instalación del módulo es totalmente análoga a la de la sección 8.2.2.2, aunque en este caso

- el fichero `.tar.gz` se llama `ClusterServer.tar.gz`
- el *script* `build.py` genera el fichero `clusterServer.sh`.

8.2.4.2. Configuración del módulo de *CygnusCloud*

Si intentamos arrancar el demonio del servidor de *cluster* sin haber inicializado previamente su fichero de configuración, se generará un mensaje de error. El fichero de configuración de este demonio, `ClusterServer_settings.conf`, se encuentra en la raíz del directorio de instalación. Su contenido inicial aparece en la figura 8.3.

Debemos recordar que, tras inicializar el contenido de este fichero, debemos borrar o comentar la sección `Uninitialized file`.

```
# Comment or delete the following line after initializing this file
[Uninitialized file]
[Database configuration]
mysqlRootsPassword =
dbUser = cygnuscloud
dbUserPassword = cygnuscloud
[Network configuration]
useSSL = False
certificatePath = /home/luis/Infraestructura/Certificates
listenningPort = 9000
[Load balancer configuration]
# The simple load balancing algorithm is no longer supported
loadBalancingAlgorithm = penalty-based
vCPUsWeight = 0.2
vCPUsExcessThreshold = 0.2
ramWeight = 0.3
storageSpaceWeight = 0.2
temporarySpaceWeight = 0.3
[Other settings]
vmBootTimeout = 120
averageCompressionRatio = 0.5
statusUpdateInterval = 5
[Image repository connection data]
imageRepositoryIP = 192.168.0.5
imageRepositoryPort = 3000
```

FIGURA 8.3: Contenido inicial del fichero `ClusterServer_settings.conf`

En primer lugar, la sección `Database configuration` especifica la información que se utilizará para configurar la base de datos del servidor de *cluster*: la contraseña del usuario *root* y el nombre y la contraseña del usuario que empleará el demonio.

Por otra parte, la sección `Network configuration` especifica la configuración de la conexión de control. En ella, se indicará si se desea cifrar el tráfico de esta conexión, la ruta de los certificados `server.crt` y `server.key` (sólo se utilizará cuando se cifra el tráfico) y el puerto asociado a la conexión de control.

Por otro lado, la sección `Load balancer configuration` especifica los parámetros de configuración del algoritmo de balanceado de carga.

Asimismo, la sección `Other settings` especifica

- el tiempo máximo que se esperará antes de generar un error de arranque de máquina virtual (en segundos).

- el ratio de compresión medio de las imágenes de disco. Este parámetro se utilizará para reservar espacio en disco en los servidores de máquinas virtuales y en el repositorio de imágenes en las operaciones de despliegue, creación, borrado y edición de imágenes.
- el intervalo de actualización del estado de la infraestructura (en segundos).

Finalmente, la sección `Image repository connection data` especifica los datos de conexión al repositorio de imágenes.

8.2.5. Servidor web

8.2.5.1. Instalación del módulo de *CygnusCloud*

Al igual que en los casos anteriores, la instalación del servidor web es análoga a la de la sección 8.2.2.2, aunque en este caso

- el fichero `.tar.gz` se llama `WebServer.tar.gz`, y
- el *script* `build.py` genera dos ficheros, `webServerEndpoint.sh` y `webServer.sh`

Para arrancar el servidor web, hay que ejecutar cada uno de ellos en un proceso independiente. El primero, `webServerEndpoint.sh`, lanza el demonio que comunica la aplicación `web2py` y el servidor de *cluster* (proceso *endpoint*), y el segundo lanza el servidor web que aloja la aplicación `web2py`.

8.2.5.2. Configuración del módulo de *CygnusCloud*

Si intentamos arrancar el proceso *endpoint* sin haber inicializado previamente su fichero de configuración, se generará un mensaje de error. El fichero de configuración de este demonio, `Endpoint_settings.conf`, se encuentra en la raíz del directorio de instalación. Su contenido inicial aparece en la figura 8.4.

Debemos recordar que, tras inicializar el contenido de este fichero, debemos borrar o comentar la sección `Uninitialized file`.

```
# Comment or delete the following line after initializing this file
[Uninitialized file]
[Database configuration]
mysqlRootsPassword =
websiteUser = website_user
websiteUserPassword = CygnusCloud
endpointUser = endpoint_user
endpointUserPassword = CygnusCloud
[Network configuration]
useSSL = False
certificatePath = /home/luis/Infraestructura/Certificates
[Cluster server connection data]
clusterServerIP = 192.168.0.5
clusterServerListenningPort = 9000
[Other settings]
statusDBUpdateInterval = 2
minCommandInterval = 1
commandTimeout = 300
commandTimeoutCheckInterval = 10
```

FIGURA 8.4: Contenido inicial del fichero `Endpoint_settings.conf`

En primer lugar, la sección `Database configuration` especifica la información que se empleará para configurar las dos bases de datos que utilizará la aplicación web para interactuar con la infraestructura:

- la contraseña del usuario *root*,
- el nombre y la contraseña del usuario que empleará el demonio, y
- el nombre y la contraseña del usuario que empleará la aplicación web.

Por otra parte, la sección `Network configuration` especifica la configuración básica de la conexión de control. En ella, se indicará si se desea cifrar el tráfico de esta conexión y la ruta de los certificados `server.crt` y `server.key` (sólo se utilizará cuando se cifra el tráfico).

Por otro lado, la sección `Cluster server connection data` especifica los parámetros de conexión al servidor de *cluster*.

Finalmente, la sección `Other settings` especifica

- el intervalo de actualización de las bases de datos (en segundos)
- el intervalo de tiempo que debe separar dos peticiones consecutivas del mismo usuario (en segundos)
- el tiempo máximo que tardarán todos los comandos en ejecutarse (en segundos), y
- el intervalo de espera del hilo de monitorización de comandos.

8.3. Creación de imágenes base

Para poder utilizar el sistema, es necesario disponer de un conjunto de configuraciones base, que se utilizarán para crear el resto de configuraciones de la infraestructura.

En esta sección, mostraremos cómo crearlas de forma sencilla utilizando `virt-manager` e importarlas en *CygnusCloud*. Naturalmente, también es posible importar a *CygnusCloud* imágenes creadas manualmente. No obstante, por claridad, en este manual *no* describiremos el proceso de creación manual de una imagen.

8.3.1. Creación de nuevas imágenes en `virt-manager`

8.3.1.1. Pasos básicos

Antes de utilizar `virt-manager`, es necesario instalarlo. Asumiendo que utilizamos Ubuntu 12.04, basta con ejecutar la orden

```
sudo apt-get install virt-manager
```

Acto seguido, lo ejecutamos. Aparecerá la pantalla inicial de la figura 8.5.

Para empezar, hacemos clic sobre el botón Create a new virtual machine. Aparecerá la primera ventana de la figura 8.7. En todos los casos, avanzamos a la ventana siguiente haciendo clic sobre Forward.

Los pasos a seguir son estos:

1. introducimos el nombre de la máquina virtual.
2. aparecerá la segunda ventana de la figura 8.7, en la que tendremos que seleccionar
   - la ubicación del medio de instalación del sistema operativo
   - el tipo del sistema operativo (Windows, Linux,...) y su versión

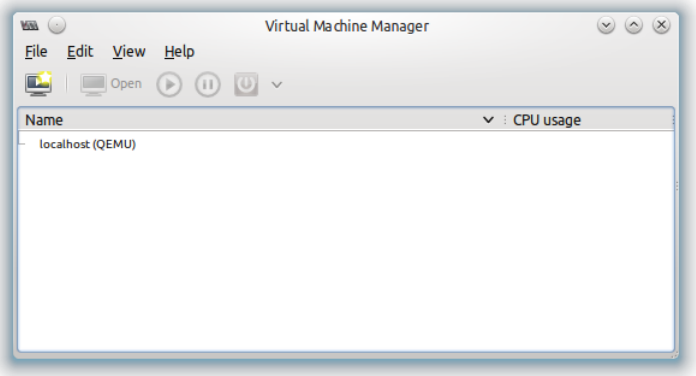

FIGURA 8.5: Pantalla inicial de `virt-manager`

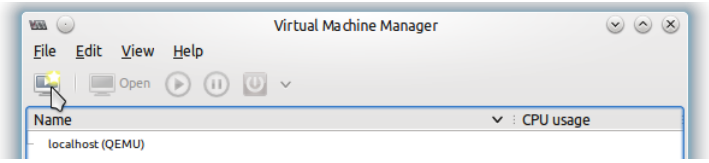

FIGURA 8.6: Ubicación del botón create a new virtual machine

3. aparecerá la primera ventana de la figura 8.8, en la que tendremos que introducir el tamaño de la RAM y el número de CPUs virtuales de la máquina.

4. desmarcamos la casilla Enable storage for this virtual machine.

5. en la ventana final del asistente, que aparece en la figura 8.9, marcamos la casilla Customize configuration before install, y hacemos clic en Finish.

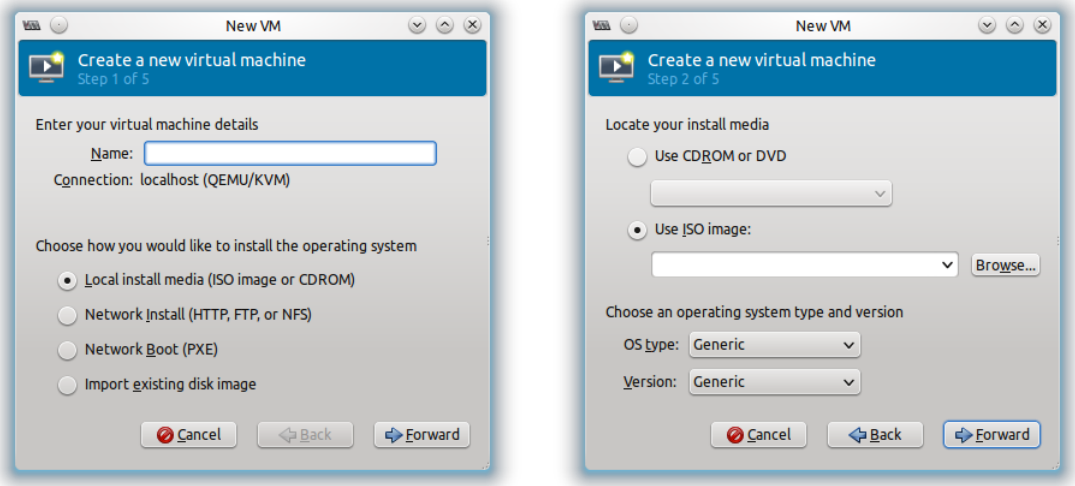

FIGURA 8.7: Pasos 1 y 2 del asistente de creación de máquinas virtuales

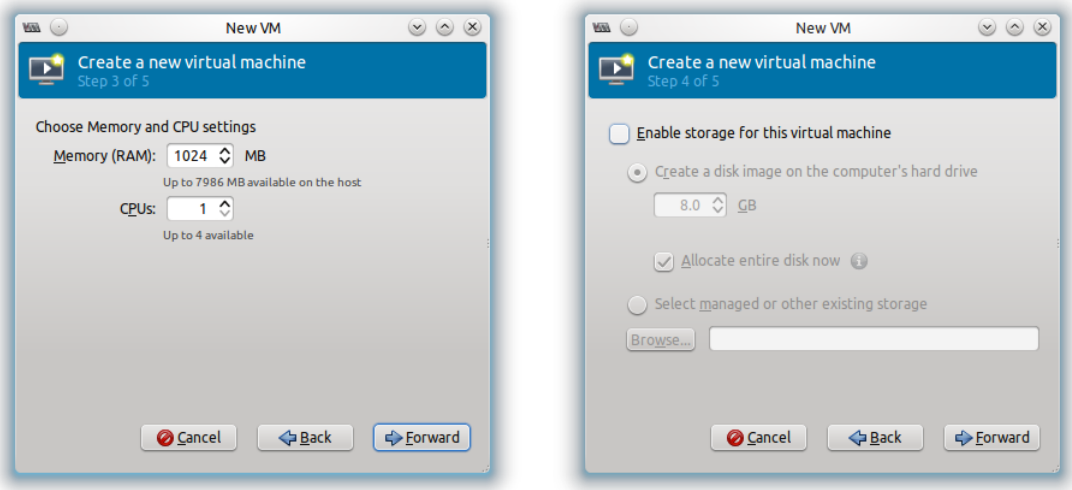

FIGURA 8.8: Pasos 3 y 4 del asistente de creación de máquinas virtuales

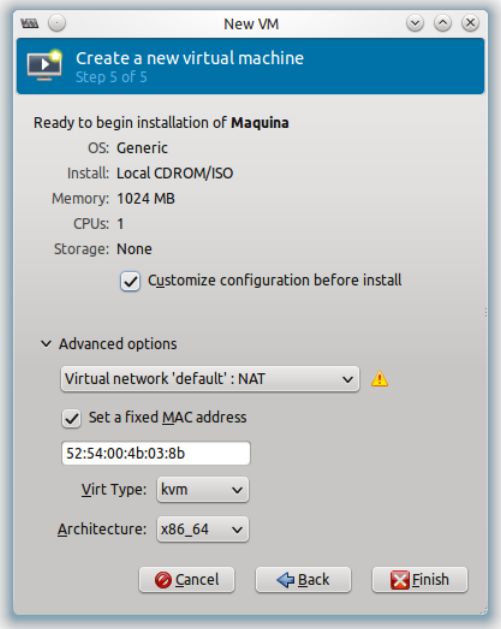

FIGURA 8.9: Paso 5 del asistente de creación de máquinas virtuales

8.3.1.2. Configuración de la máquina virtual

Tras los pasos de la sección 8.3.1.1, `virt-manager` ya ha generado casi todo el fichero de definición de la máquina. No obstante, todavía debemos configurar

- las características de la CPU de la máquina virtual
- los discos duros virtuales
- los periféricos de entrada
- la tarjeta de vídeo

Cuando se cierra el asistente, aparecerá la ventana de la figura 8.10.

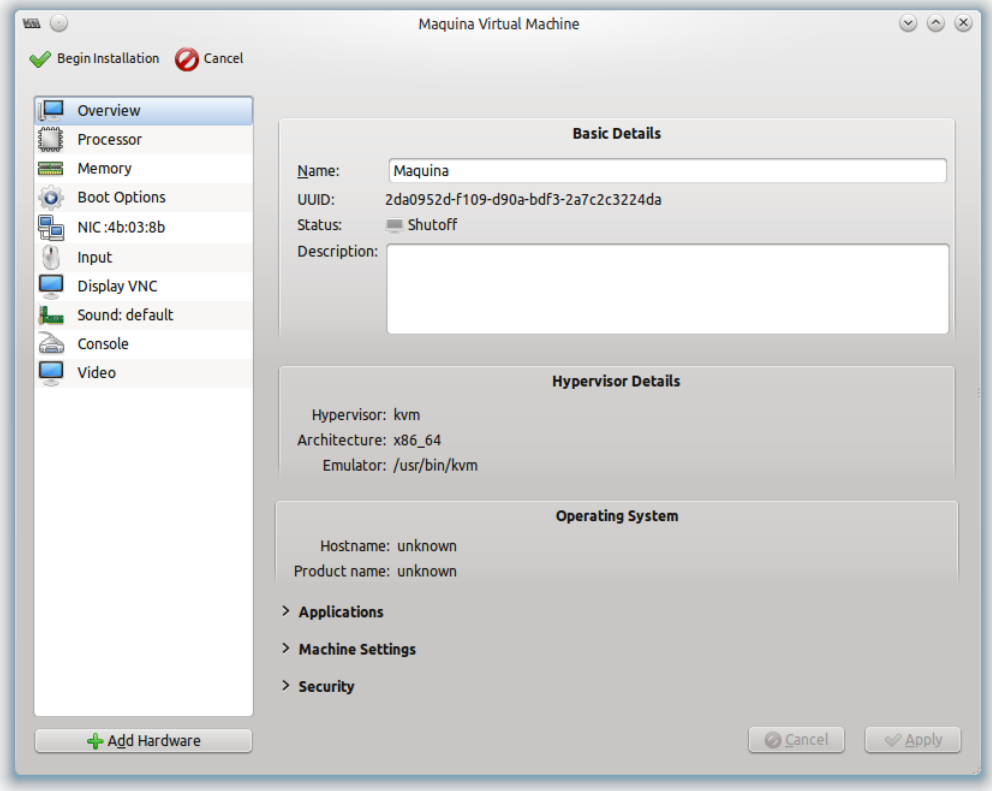

FIGURA 8.10: Diálogo de configuración de la máquina virtual

Empezaremos configurando la CPU de la máquina. Para ello, hacemos clic sobre Processor. Tras expandir el apartado Configuration, seleccionamos Core2Duo como modelo. Todo debería quedar configurado como en la figura 8.11.

A continuación, crearemos las imágenes de disco `qcow2` que utilizará la máquina. Tenemos que crear dos:

- una que contendrá el sistema operativo y los programas instalados
- otra que contendrá los datos del usuario y el fichero de paginación

Para crearlas, hacemos clic sobre el botón Add Hardware y seleccionamos Storage. El diálogo que aparece se muestra en la figura 8.12.

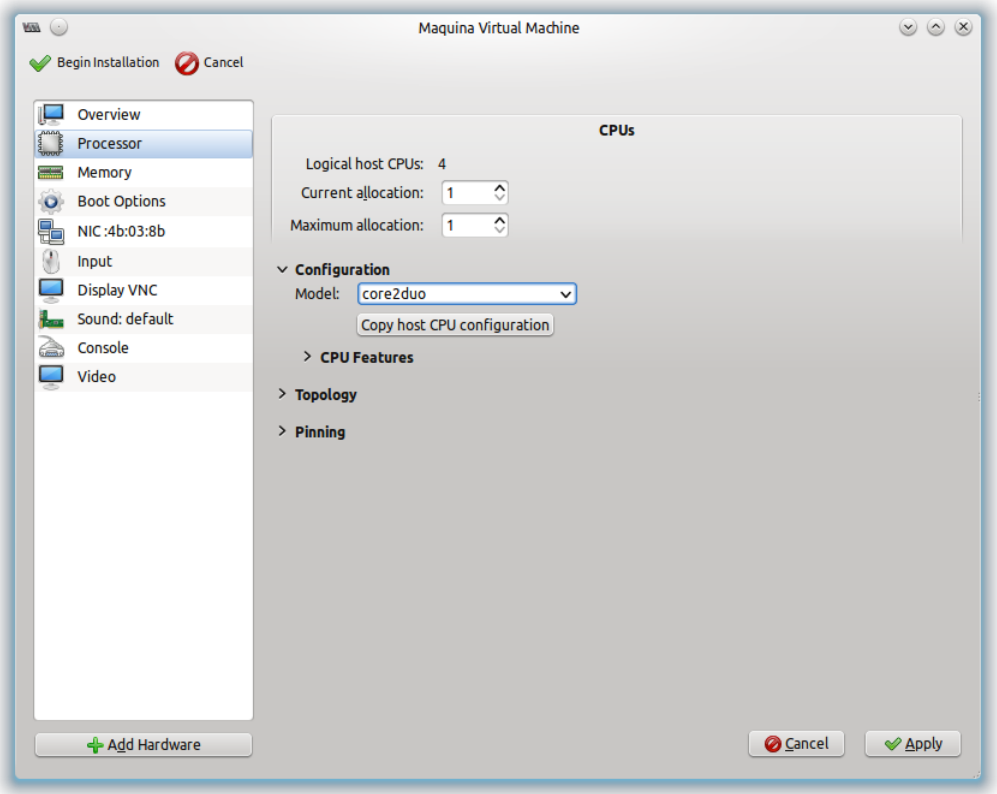

FIGURA 8.11: Configuración del procesador de la máquina virtual

FIGURA 8.12: Creación de las imágenes de disco

A la hora de especificar el tamaño de ambas, debemos

- escoger una familiar de máquinas virtuales, y

- seleccionar Virtio disk en Device type y `qcow2` en Storage format.

Por otra parte, para configurar la tarjeta de vídeo hacemos clic sobre Video, y en Model seleccionamos vga. Esto se muestra en la figura 8.13.

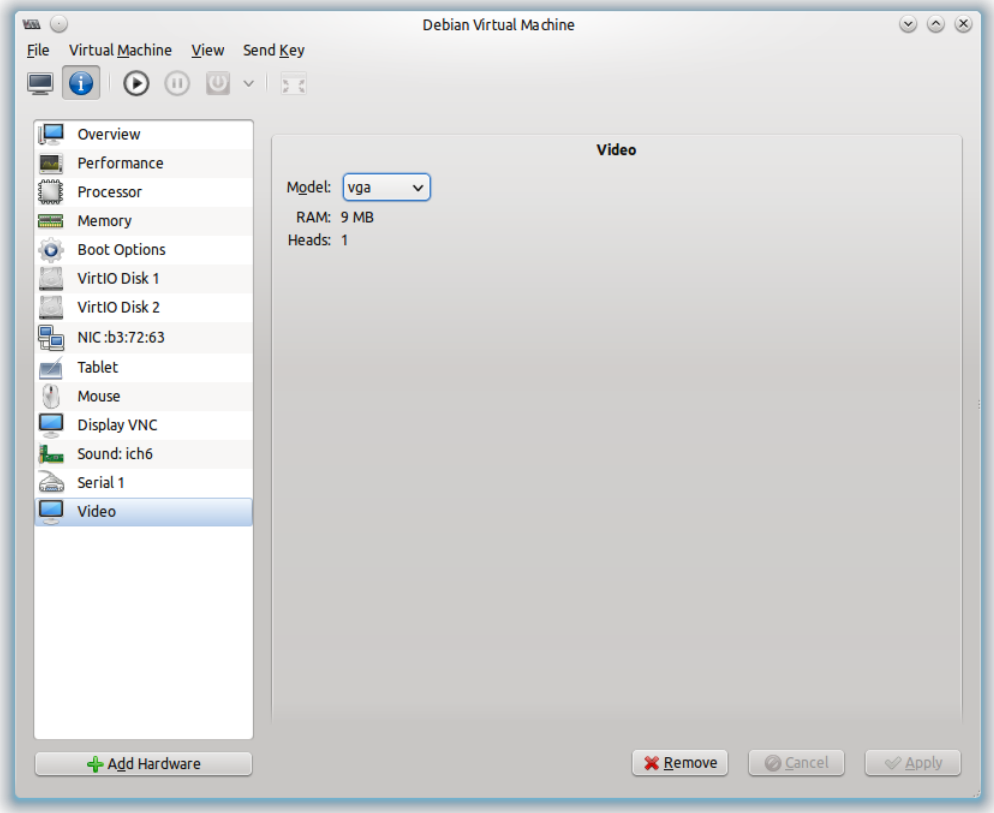

FIGURA 8.13: Configuración de la tarjeta de vídeo

Importante
Debido a un *bug* en el cargador de arranque, todas las distribuciones derivadas de Debian 6 (*squeeze*) no pueden arrancar con la tarjeta de vídeo vmvga. En estos casos, es necesario seleccionar la tarjeta de vídeo vga.

Lo último que nos falta es añadir una tableta gráfica para que el servidor VNC siga adecuadamente los movimientos del ratón. Para ello, hacemos clic sobre Add Hardware, seleccionamos Input, seleccionamos EvTouch USB Graphics Tablet y hacemos clic sobre Finish.

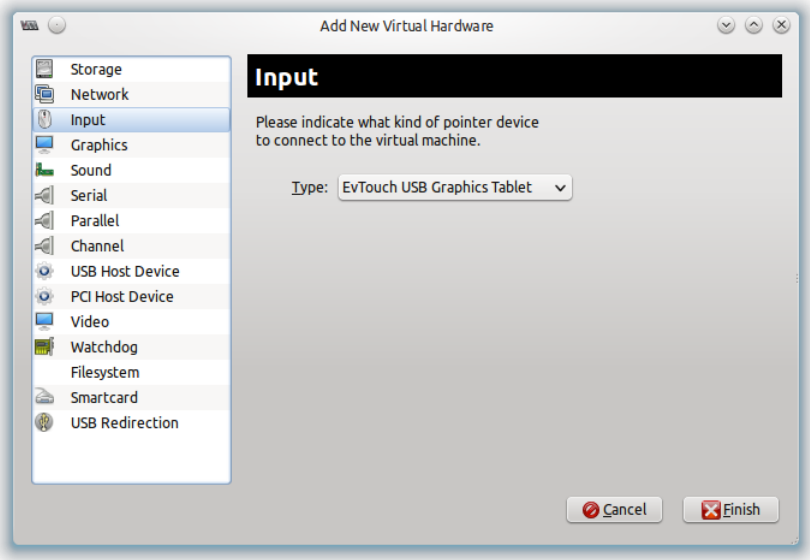

FIGURA 8.14: Configuración de los periféricos de entrada

Para empezar la instalación del sistema operativo, hacemos clic sobre Begin installation.

Importante
Para poder instalar un sistema operativo *Windows*, hay que suministrar al instalador los drivers VirtIO. Mostraremos cómo hacer esto en la siguiente sección.

8.3.1.3. Instalación de los *drivers* en *Windows*

Para que el instalador pueda acceder a los discos duros virtuales, hay que suministrarle los controladores VirtIO. Los pasos a seguir son estos:

1. Descargamos la imagen ISO correspondiente a la última versión desde

   http://alt.fedoraproject.org/pub/alt/virtio-win/latest/images/bin/.

2. A la hora de configurar la máquina, creamos otra unidad CD-ROM IDE. Para ello, basta con hacer clic sobre Add hardware, seleccionar Storage y

   a) marcar la casilla Select managed or other existing storage, e introducir la ruta de la imagen ISO que hemos descargado.

   b) seleccionar IDE cdrom en Device type.

En ocasiones, `virt-manager` selecciona la nueva unidad de CD-ROM (y no la que tiene el medio de instalación de *Windows*) para arrancar. Si eso ocurre, basta con volver a abrir el diálogo de configuración de la máquina (hacemos clic derecho sobre la máquina en la ventana principal de `virt-manager`, seleccionamos Open y accedemos al menú View ▷ Details de la nueva ventana que aparece), seleccionar Boot options, marcar la casilla Enable boot menu y seleccionar la unidad correcta al arrancar la máquina.

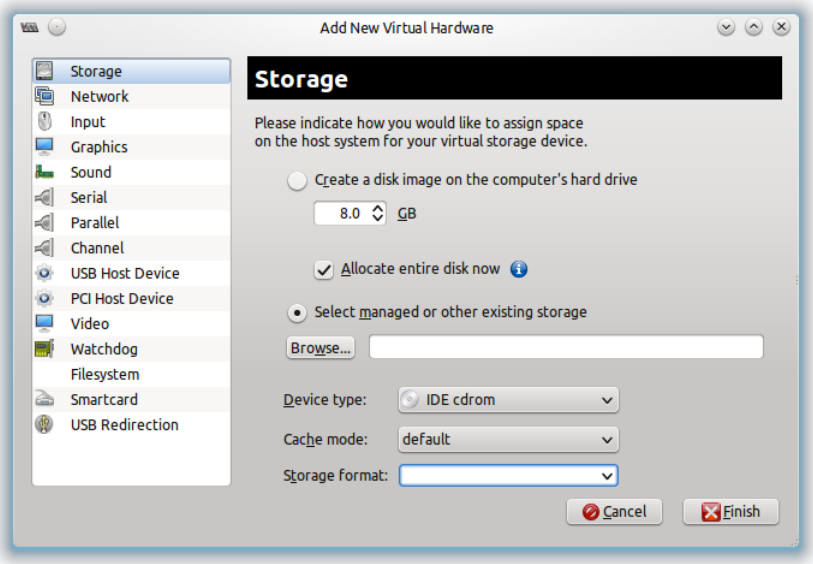

FIGURA 8.15: Configuración del CD-ROM con los drivers VirtIO

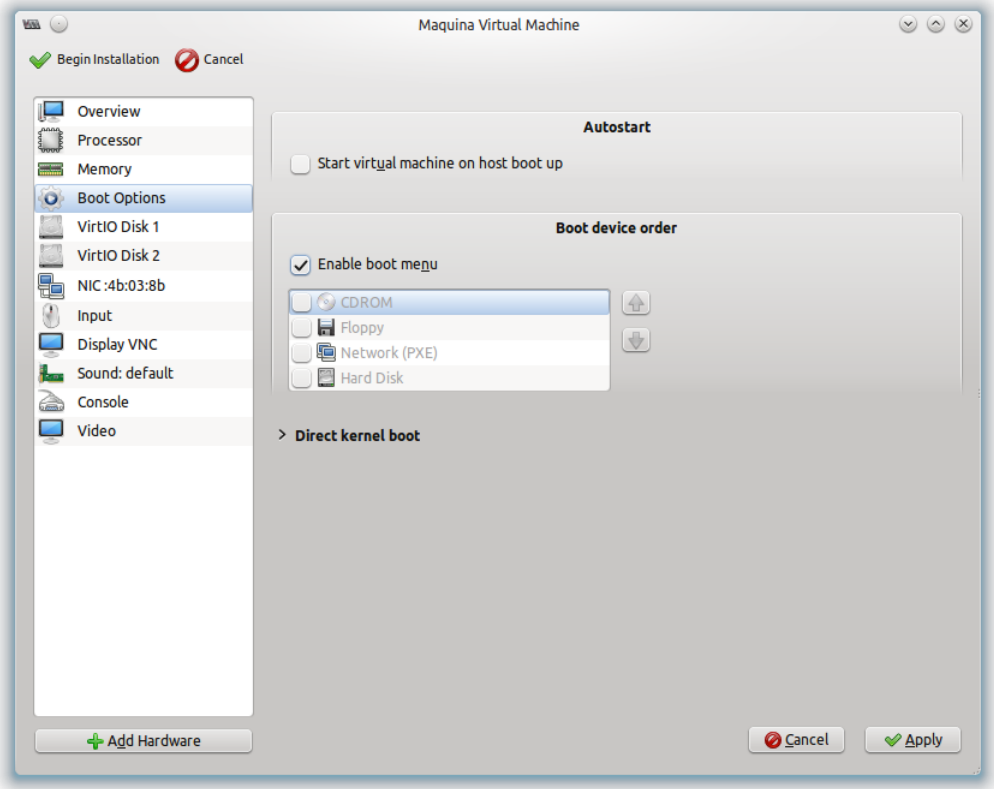

FIGURA 8.16: Corrección de los problemas de arranque en instalaciones *Windows*

8.3.2. Importar imágenes existentes a `virt-manager`

Siempre que una imagen de disco tenga instalado un sistema *Linux* y uno de los tamaños asociados a las imágenes del sistema operativo de las familias de máquinas virtuales *Linux*, podrá utilizarse en *CygnusCloud*. Los pasos a seguir son los siguientes:

1. se convierte la imagen de disco a un formato soportado por QEMU (`raw`, `vmdk` , `qcow2` o `qcow2`). Por ejemplo, en el caso de una imagen de *VirtualBox*, habría que ejecutar el comando

   ```
   VBoxManage clonehd <fichero fuente>.vdi <fichero destino>.vmdk --format VMDK
   ```

2. se convierte la imagen al formato `qcow2`. Para ello, ejecutaríamos la orden

   ```
   qemu-img convert <fichero fuente>.vmdk -O qcow2 <fichero destino>.qcow2
   ```

3. en lugar de crear una nueva imagen para el sistema operativo, en el diálogo se importa la existente seleccionando la casilla Select managed or other existing storage e indicando su ruta.

4. se crea una imagen de disco para almacenar los datos del usuario y el fichero de paginación.

5. tras arrancar la máquina virtual, se mueve el fichero de paginación y el directorio `/home` a la imagen de disco correspondiente.

Importante: este procedimiento no funcionará con imágenes de disco que utilicen el sistema operativo *Windows*.

8.3.3. Uso de *Samba* para configurar las imágenes

Para configurar el entorno de la máquina virtual, no sólo es posible descargar *software* de internet: también es posible, utilizando *Samba*, acceder a los recursos de la máquina anfitrión o de cualquier otro PC conectado a la misma red de área local que la máquina anfitrión.

En esta sección mostraremos cómo instalar y configurar *Samba* para que esto sea posible. Distinguiremos cuatro casos:

- el PC y la máquina virtual tienen instalado un sistema operativo *Windows*. Llamaremos a este caso **compartición *Windows* a *Windows***.
- el PC tiene instalado un sistema operativo *Windows*, y la máquina virtual tiene instalado un sistema operativo *Linux*. Llamaremos a este caso **compartición *Windows* a *Linux***.
- el PC tiene instalado un sistema operativo *Linux*, y la máquina virtual tiene instalado un sistema operativo *Windows*. Llamaremos a este caso **compartición *Linux* a *Windows***.
- el PC tiene instalado un sistema operativo *Linux*, y la máquina virtual tiene instalado un sistema operativo *Linux*. Llamaremos a este caso **compartición *Linux* a *Linux***.

Antes de empezar, mostraremos qué hace falta instalar en las imágenes *Linux* que utilicen distribuciones derivadas de Debian o Ubuntu. En las imágenes *Windows*, no es necesario instalar nada.

8.3.3.1. Prerrequisitos (sólo en *Linux*)

8.3.3.1.1. Máquina virtual

Si la máquina virtual utiliza un sistema operativo *Linux*, habrá que instalar todo lo necesario para que se convierta en un cliente *Samba*. Para ello, basta con ejecutar la orden

```
sudo apt-get install smbclient smbfs
```

8.3.3.1.2. PC

Si el PC tiene instalado un sistema operativo *Linux*, habrá que instalar todo lo necesario para que se convierta en un servidor *Samba*. Para ello, ejecutaremos la orden

```
sudo apt-get install samba
```

Después, editamos el fichero `/etc/samba/smb.conf`, descomentando las líneas

```
security = user
```

y

```
workgroup = WORKGROUP
```

Para terminar, definimos una contraseña para nuestro usuario. *Samba* no nos obliga a utilizar nuestro propio usuario: podemos crear a propósito un usuario para *Samba*. De todas formas, lo más sencillo es utilizar el que ya tenemos. El comando que tenemos que escribir es

```
sudo smbpasswd -a <nuestro nombre de usuario>
```

Esta orden nos pedirá la nueva contraseña.

Importante: si omitimos este paso, no será posible montar ninguna carpeta compartida. Este error es muy difícil de detectar.

8.3.3.2. Compartición de *Windows* a *Windows*

8.3.3.2.1. Configuración del PC

Para compartir una carpeta en *Windows*, basta con seguir estos pasos:

1. hacemos clic derecho sobre la carpeta que queremos compartir y seleccionamos Propiedades.
2. vamos a la pestaña Compartir y hacemos clic sobre el botón Uso compartido avanzado
3. marcamos la casilla Compartir esta carpeta. Por defecto, sólo se permite leer de ella. En la parte superior del diálogo, aparecerá un nombre de la forma

   ```
   \\nombre-del-pc\nombre-del-recurso
   ```

   Debemos recordar el nombre del recurso: nos hará falta más adelante.
4. si queremos escribir en la carpeta desde otra máquina, basta con hacer clic sobre Permisos y marcar la casilla Control total.

8.3.3.2.2. Configuración de la máquina virtual

Para conectarnos a la carpeta compartida desde la máquina virtual, seguimos estos pasos:

1. hacemos clic sobre Equipo, y seleccionamos Conectar a unidad de red.
2. en el diálogo que aparece, metemos este nombre para la máquina:

   ```
   \\ip-del-pc\nombre-del-recurso
   ```
3. confirmamos los cambios. A partir de este momento, la carpeta compartida aparecerá montada en Equipo como una unidad de red.

8.3.3.3. Compartición de *Linux* a *Linux*

8.3.3.3.1. Configuración del PC

Para compartir la carpeta, seguimos los siguientes pasos:

1. añadimos una entrada como la siguiente *al final* del fichero `/etc/samba/smb.conf`:

```
[nombre-del-recurso]
  comment = <comentario>
  read only = yes | no
  guest ok = yes | no
  browsable = yes
  force user = <nuestro usuario>
  path = <ruta de la carpeta compartida>
```

   donde

   - `nombre-del-recurso` es el nombre que queremos asignar a la carpeta compartida (como `home`, `mis_datos`,...). Por seguridad, *no* metáis aquí espacios ni caracteres especiales.
   - `comment` contiene una descripción de la carpeta compartida. Si está vacío, no aparece nada a la derecha del símbolo `=`.
   - `read only` fija los permisos de lectura y escritura. Cuando toma el valor `yes`, no se podrá escribir en la carpeta desde la máquina virtual. Cuando toma el valor `no`, sí.
   - `guest ok` determina el comportamiento para los usuarios no autenticados. Cuando toma el valor `yes`, cualquiera podrá leer de la carpeta. Cuando toma el valor `no`, sólo los usuarios que se autentiquen podrán leer la carpeta.

2. reiniciamos el servidor *Samba*. Para ello, ejecutamos la orden

```
sudo /etc/init.d/smbd restart
```

8.3.3.3.2. Configuración de la máquina virtual

Para montar la carpeta compartida en la máquina virtual, seguimos los siguientes pasos:

1. creamos el punto de montaje. Por ejemplo, si queremos montar la carpeta en `/mnt/smbshare`, ejecutaremos el comando

```
sudo mkdir /mnt/smbshare
```

2. montamos la carpeta utilizando el comando

```
sudo mount -t cifs -o username=<nombre de usuario>,password=<contraseña>
           //ip-del-pc/nombre-del-recurso <punto de montaje>
```

3. para desconectarnos de la carpeta compartida, basta con ejecutar el comando

```
sudo umount --force <punto de montaje>
```

8.3.3.4. Compartición de *Windows* a *Linux*

Para configurar el PC, seguimos las instrucciones de la sección 8.3.3.2. Para configurar la máquina virtual, seguimos las instrucciones de la sección 8.3.3.3.

8.3.3.5. Compartición de *Linux* a *Windows*

Para configurar el PC, seguimos las instrucciones de la sección 8.3.3.3. Para configurar la máquina virtual, seguimos las instrucciones de la sección 8.3.3.2.

8.3.4. Importar imágenes a *CygnusCloud*

Una vez preparadas las imágenes de disco de las máquinas virtuales, tenemos que borrar las unidades de CD-ROM que hemos utilizado para realizar la instalación del sistema operativo, y también la tarjeta de sonido. Los pasos a seguir son los siguientes:

1. accedemos al diálogo de configuración de la máquina. Para ello, hacemos clic derecho sobre la máquina en la ventana principal de virt-manager, seleccionamos Open y accedemos al menú View ▷ Details.
2. hacemos clic derecho sobre las unidades de CD-ROM y seleccionamos Remove hardware.
3. hacemos lo mismo con la tarjeta de sonido, que aparece en la sección sound: ich6.

Una vez hecho esto,

1. creamos un directorio.
2. copiamos el fichero de definición de la máquina, ubicado en `/etc/libvirt/qemu`, a dicho directorio.
3. copiamos las dos imágenes de disco `qcow2` a dicho directorio. Acto seguido, las renombramos de la siguiente manera:

   Imagen del sistema operativo $\longrightarrow$ `OS.qcow2`

   Imagen restante $\longrightarrow$ `Data.qcow2`
4. creamos un fichero `zip` que contenga el fichero de definición y las dos imágenes de disco `qcow2` *renombradas*. Estos archivos *deben estar ubicados en la raíz del fichero comprimido*.

   Importante: la infraestructura **no** podrá utilizar la imagen cuando

   - no se respeten los nombres de los ficheros `qcow2`, incluyendo la distinción entre mayúsculas y minúsculas, o cuando
   - los tres archivos no estén en la raíz del fichero comprimido
5. subimos el fichero `.zip` al repositorio de imágenes mediante la utilidad de transferencia de imágenes.

   Importante: hay que anotar el identificador devuelto por la utilidad de conexión.
6. abrimos el fichero `cc/clusterServer/database/ClusterServerDB.sql`, siendo `cc` el directorio de instalación del servidor de *cluster*. Acto seguido, añadimos la línea

   ```
   INSERT IGNORE INTO VanillaImageFamilyOf VALUES (IDi, IDf);
   ```

   al final del mismo. `IDi` es el identificador de imagen que ha devuelto la utilidad de conexión, e `IDf` es el identificador de la familia de máquinas virtuales a la que está asociada la imagen. El fichero `.sql` contiene la definición de todas las familias de máquinas virtuales.
7. abrimos el fichero `ws/ClusterEndpoint/databases/ClusterEndpointDB.sql`, donde `ws` es el directorio de instalación del servidor web. Acto seguido, añadimos al final de ese fichero la línea

   ```
   INSERT IGNORE INTO Image VALUES
       (IDi, IDf, 'Nombre imagen', 'Descripción imagen',
           ID_osf, ID_osv, 1, 0);
   ```

   siendo `ID_osf` el identificador de la familia de sistemas operativos a la que pertenece la imagen, e `ID_osv` el identificador de la variante del sistema operativo que utiliza la imagen. Estos valores se extraen de las tablas `OSFamily` y `OSVariant`, ya rellenadas.
8. reiniciamos la infraestructura. A partir de este momento, podremos utilizar la imagen como base para crear otras nuevas.

8.4. Uso de la página web

8.4.1. Instalación y uso del *framework* web2py

La web de *CygnusCloud* se ha desarrollado utilizando el *framework* web2py. Para poder utilizar este *framework* no es necesario realizar ningún tipo de instalación adicional a la indicada en la sección 8.2.5. El fichero .tar.gz ya incluye la última versión estable de web2py (2.5.1) y la aplicación web de *CygnusCloud* integrada. No obstante, si se quiere descargar una versión diferente será necesario seguir los siguientes pasos:

1. Nos conectamos a http://www.web2py.com/init/default/download y descargamos la versión deseada. Es posible descargar versiones para *Linux*, *Windows* o Mac OS, aunque para poder ejecutar *CygnusCloud* sólo puede usarse la versión de *Linux*. También es posible elegir entre 3 tipos de descargas de cara al uso que se quiera hacer de web2py. En nuestro caso, usaremos la opción For Normal Users.

2. descomprimimos el fichero descargado y lo copiamos al directorio de instalación que queramos.

3. copiamos el directorio web2py/applications/CygnusCloud del *tarball que aportamos* al directorio de instalación de web2py.

4. copiamos los directorios bin y certificates, el script build.py y los ficheros de configuración al directorio donde se haya descomprimido la versión de web2py (que contendrá un directorio con el nombre web2py).

5. realizamos la instalación tal y como se menciona en la sección 8.2.5.

Importante
el *tarball* que hemos publicado ya incluye una versión de web2py estable y configurada. Sólo tendrán que seguir este procedimiento aquellos usuarios que quieran utilizar una versión diferente del *framework* web2py.

web2py ofrece una interfaz para el desarrollo de aplicaciones web accesible desde cualquier navegador. Tras realizar la instalación y ejecutar el script webServer.sh podremos utilizarla siguiendo estos pasos:

1. introducimos la URL http://127.0.0.1:8000/ en un navegador web.

2. Se cargará una página con el título *Welcome*. Hacemos clic sobre el botón Administrative Interface, situado a la derecha de esta ventana.

3. En la siguiente página, escribiremos la contraseña de administrador (cygnuscloud) y pulsaremos el botón de inicio de sesión.

4. Llegados a este punto nos aparece un escritorio como el que se muestra en 8.17, con el nombre de las aplicaciones que podemos gestionar desde web2py. Sobre CygnusCloud, seleccionamos la opción Manage ▷ editar.

5. se cargará una página con el título Editar aplicación "CygnusCloud". En esta página podemos ver todo el código fuente de la web de *CygnusCloud* organizado en secciones según su finalidad: modelos, controladores, vistas, lenguajes, ficheros estáticos, módulos secundarios, ficheros privados y *plugins*.

6. A la izquierda de cada fichero hay un botón editar que nos permite modificar el fichero de código correspondiente. Igualmente si queremos visualizar el contenido del fichero sin editarlo podemos pulsar directamente sobre su nombre.

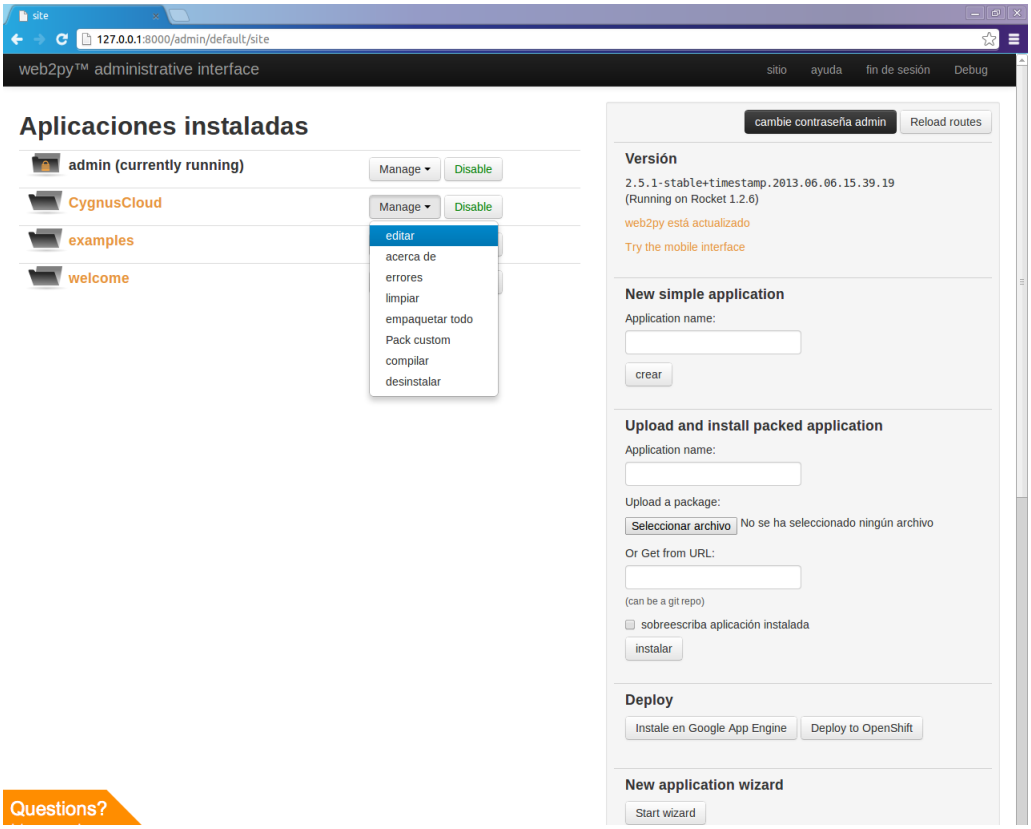

FIGURA 8.17: Interfaz administrativa de web2py

8.4.2. Uso de la web

8.4.2.1. Páginas de acceso público

Tras instalar el *tarball* y ejecutar los *scripts* de arranque del servidor web y del endpoint, podemos acceder a la página de inicio de sesión de *CygnusCloud* introduciendo la URL `http://127.0.0.1:8000/CygnusCloud/main/login` en cualquier navegador web. La figura8.18 muestra una captura de pantalla de esta página.

En ella, debemos introducir el correo electrónico y la contraseña del usuario que va a iniciar sesión.

Por defecto, existen tres usuarios de prueba registrados en el sistema:

- un estudiante con correo electrónico `Student1@ucm.es` y contraseña `1234`,
- un profesor con correo electrónico `Teacher01@ucm.es` y contraseña `1234`, y
- un administrador con correo electrónico `Admin1@ucm.es` y contraseña `1234`.

Además, desde esta página también es posible

- acceder a diferentes direcciones institucionales tales como el campus virtual o la página web de la UCM, y
- visualizar los últimos *tweets* de la cuenta `@CygnusCloud`

Una vez iniciada la sesión, independientemente del tipo de usuario que la inicie, en la parte superior derecha de la página aparece el nombre del usuario cuya sesión se ha iniciado y una opción `Salir` que permite cerrar la sesión actual. Cuando un usuario cierre la sesión, regresará a la página de inicio.

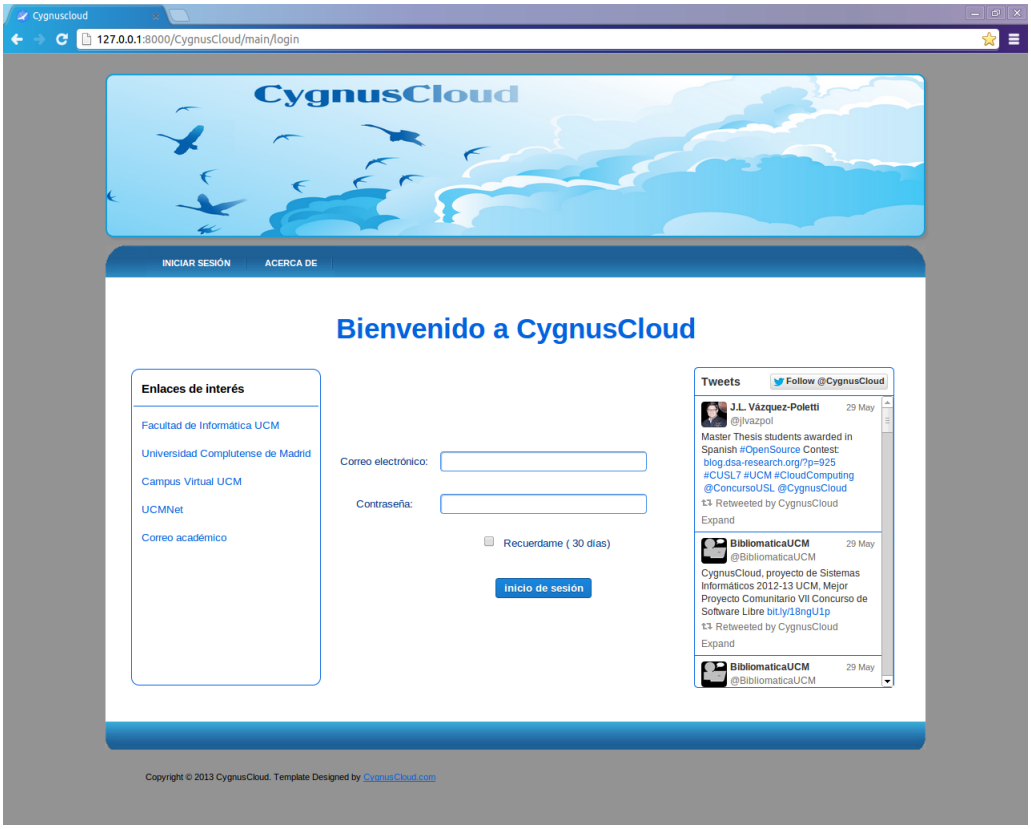

FIGURA 8.18: Página de inicio de *CygnusCloud*

Desde la barra de menú de acceso público, también es posible acceder a la página de `Acerca de`, en la cual se incluye un breve resumen sobre que es *CygnusCloud* y para que sirve. También aparece en esta página información de interés sobre el desarrollo de *CygnusCloud*, tales como entrevistas, notas de prensas o premios recibidos.

8.4.2.2. Páginas accesibles para los estudiantes

Tras iniciar sesión como un estudiante, la aplicación web nos muestra una página como la de la figura 8.19, en la cual aparecen todas las asignaturas matriculadas por el estudiante. Para cada asignatura, se indica las máquinas creadas por el profesor de dicha asignatura.

El estudiante puede seleccionar cualquiera de las máquinas que se le muestran. Cuando se selecciona, aparece una breve descripción sobre dicha máquina y un botón `Arrancar`.

Una vez el estudiante ha seleccionado la máquina que quiere arrancar y ha pulsado sobre el botón `Arrancar`, se abre una nueva pestaña en la cual comienza a ejecutarse el escritorio de trabajo de la máquina arrancada. La figura 8.20 muestra la página con el escritorio de trabajo arrancado.

En esta página, además del escritorio de trabajo, se incluye un par de botones que permiten enviar una señal de apagado a la máquina y poner el escritorio en pantalla completa.

Además, el estudiante podrá gestionar sus máquinas arrancadas por medio de las páginas de detención y apertura de máquinas arrancadas (Máquinas arrancadas ▷ Detener máquina y Máquinas arrancadas ▷ Abrir maquina). La forma de proceder en ambas páginas es similar. Una vez haya accedido a alguna de las páginas, se le muestra una lista con todas las máquinas que tiene arrancadas. Al seleccionar cualquiera de ellas aparece la descripción de la máquina y los botones con las acciones que puede realizar sobre dicha máquina. En el caso de la página de detención, se muestra un botón que permite detener la máquina y otro que permite forzar el reinicio. En el caso de la página de apertura, aparece un único botón que permite abrir una nueva pestaña con el escritorio de alguna de las máquinas en ejecución. La figura 8.21 muestra una captura de la página de detención.

Importante

- para poder arrancar una máquina virtual, un administrador debe haber arrancado previamente al menos uno de los servidores de máquinas virtuales en los que se encuentra.
- para que el rendimiento del visor de escritorio remoto sea adecuado, es imprescindible utilizar la última versión de *Google Chrome, Mozilla Firefox, Internet Explorer* o *Safari*.

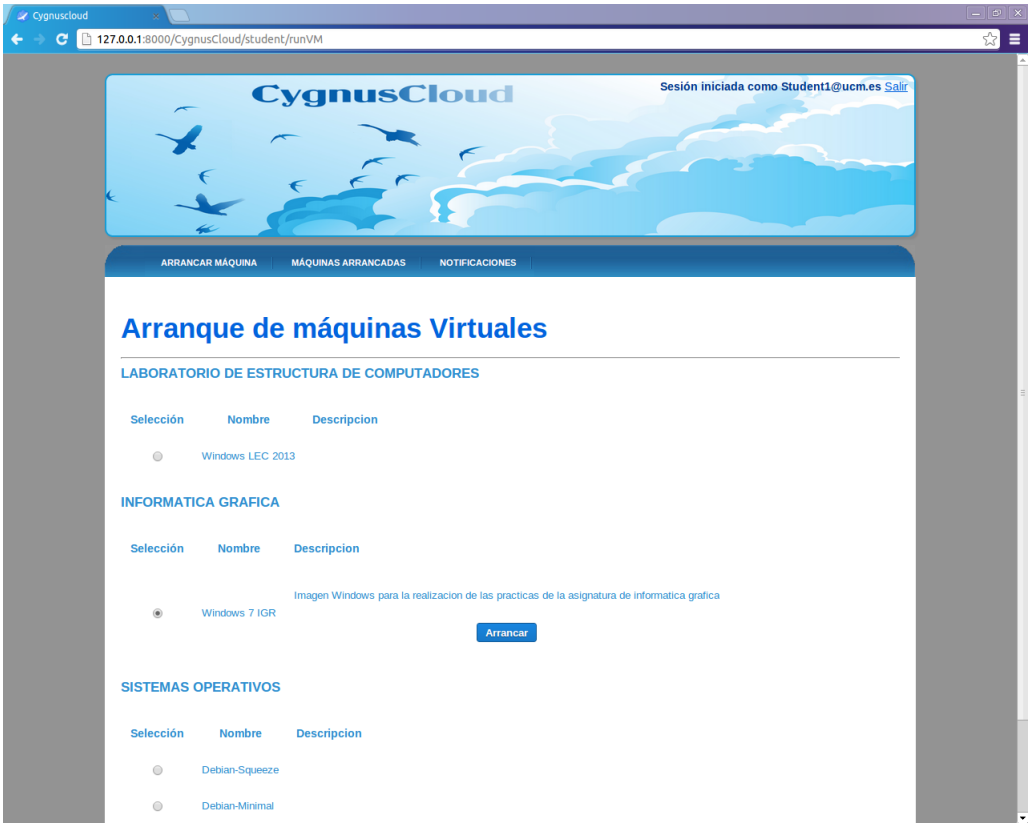

FIGURA 8.19: Página de arranque de máquinas virtuales por un estudiante

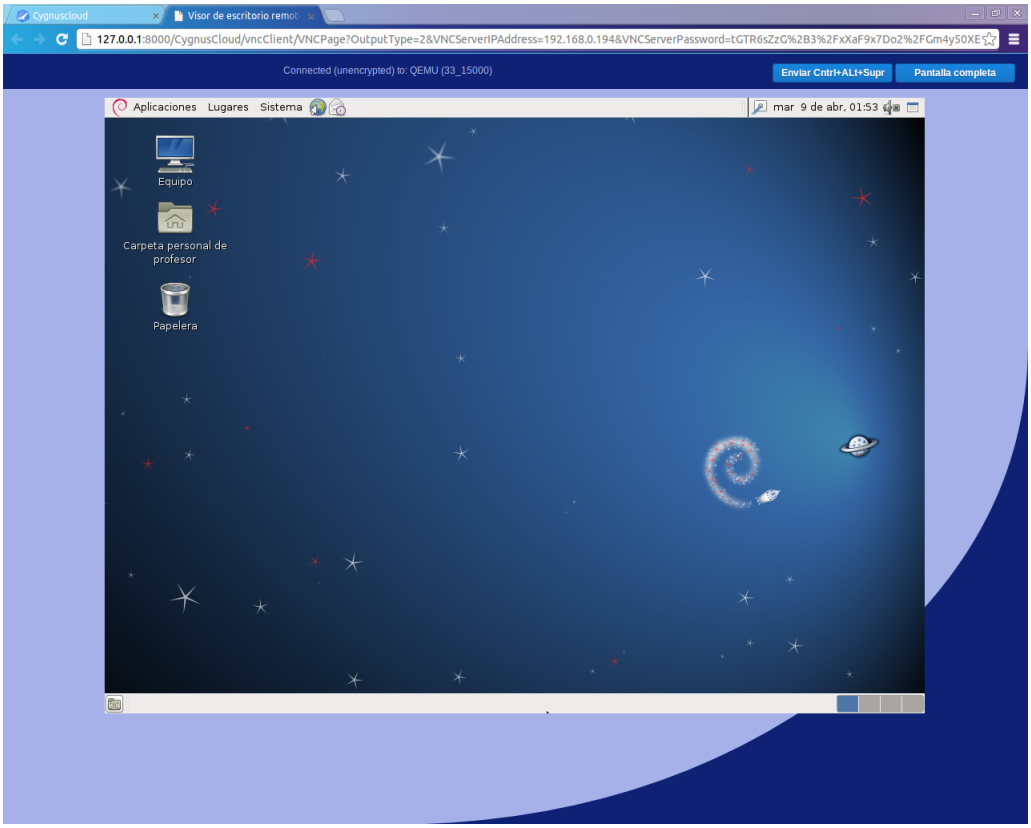

FIGURA 8.20: Página con el escritorio remoto de una máquina virtual arrancada por un estudiante

FIGURA 8.21: Página de detención de máquinas virtuales en ejecución para los estudiantes

8.4.2.3. Páginas accesibles para los profesores

Una vez que el profesor ha iniciado sesión, la primera página que se le muestra es la página de arranque de máquinas virtuales. El manejo de esta página, al igual que las páginas de deten-

ción y apertura de máquinas virtuales arrancadas (`Máquinas arrancadas ▷Detener máquina` y `Máquinas arrancadas ▷Abrir maquina`) es similar al explicado en la sección de estudiantes, por lo que no se repetirá en esta sección.

La página de creación de imágenes (`Crear y editar ▷ Crear nueva máquina`) permite a los profesores crear nuevas imágenes sobre las cuales instalarán las herramientas necesarias para poder ser utilizadas por los estudiantes matriculados en una de las asignaturas que imparten. Para llevar a cabo la creación de una imagen, el profesor debe indicar un nombre y una descripción que se les mostrará a los estudiantes. Además, debe seleccionar una imagen base sobre la cual instalará las herramientas necesarias. Las imágenes bases se distinguen por el sistema operativo instalado en ellas y los recursos que se le proporcionan a cada una. Es posible filtrar la lista de imágenes bases por su sistema operativo. Una vez rellenados todos los campos, el profesor debe pulsar sobre el botón Crear máquina virtual para enviar la petición de creación.

La figura 8.22 muestra un ejemplo de creación de una máquina virtual *Windows*.

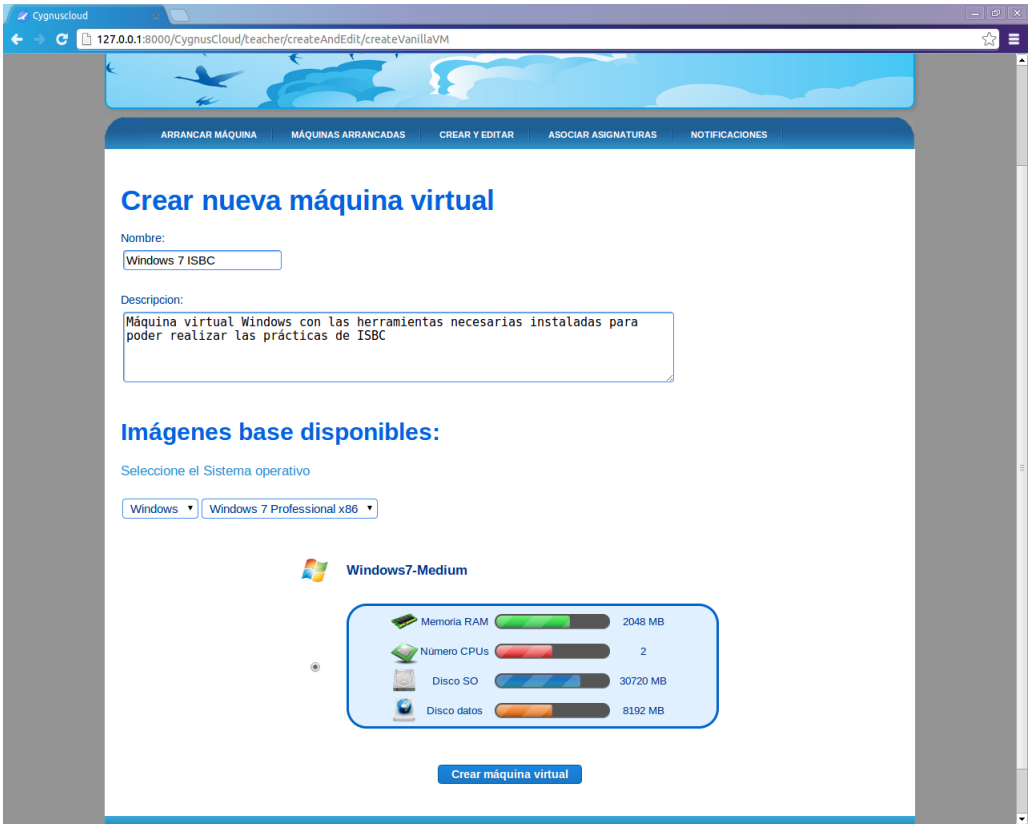

FIGURA 8.22: Página de creación de máquinas virtuales para los profesores

La página de edición de imágenes (`Crear y editar ▷ Editar máquina`) permite a los profesores editar sus máquinas ya creadas.

En esta página aparecen dos tablas donde se muestran el conjunto de todas las máquinas asociadas a las asignaturas impartidas por el profesor. La primera tabla contiene el conjunto de máquinas detenidas.

Una máquina detenida puede ser:

- Máquina que aun no se ha comenzado a editar
- Máquina que se ha editado, se ha detenido, pero aun no se han aplicado sus cambios.

La segunda tabla muestra el conjunto de máquinas en ejecución. En esta página una máquina en ejecución es:

- Máquina que se ha ejecutado para ser editada

Según el estado en el que se encuentre una máquina, al seleccionarla, se mostrará unos botones u otros.

Por ejemplo, la figura 8.23 muestra la página de edición de un profesor. En ella podemos observar las dos tablas. En la primera se ha seleccionado una máquina que ya se ha editado anteriormente y ha sido detenida. Las opciones que se le ofrece al profesor en este caso son seguir editando la máquina o aplicar los cambios para hacerlos visibles a los estudiantes. En la segunda tabla se ha seleccionado una máquina en edición que aun no se ha detenido. Las dos posibilidades que tiene en este caso el profesor son abrir la máquina en una pestaña nueva o detenerla.

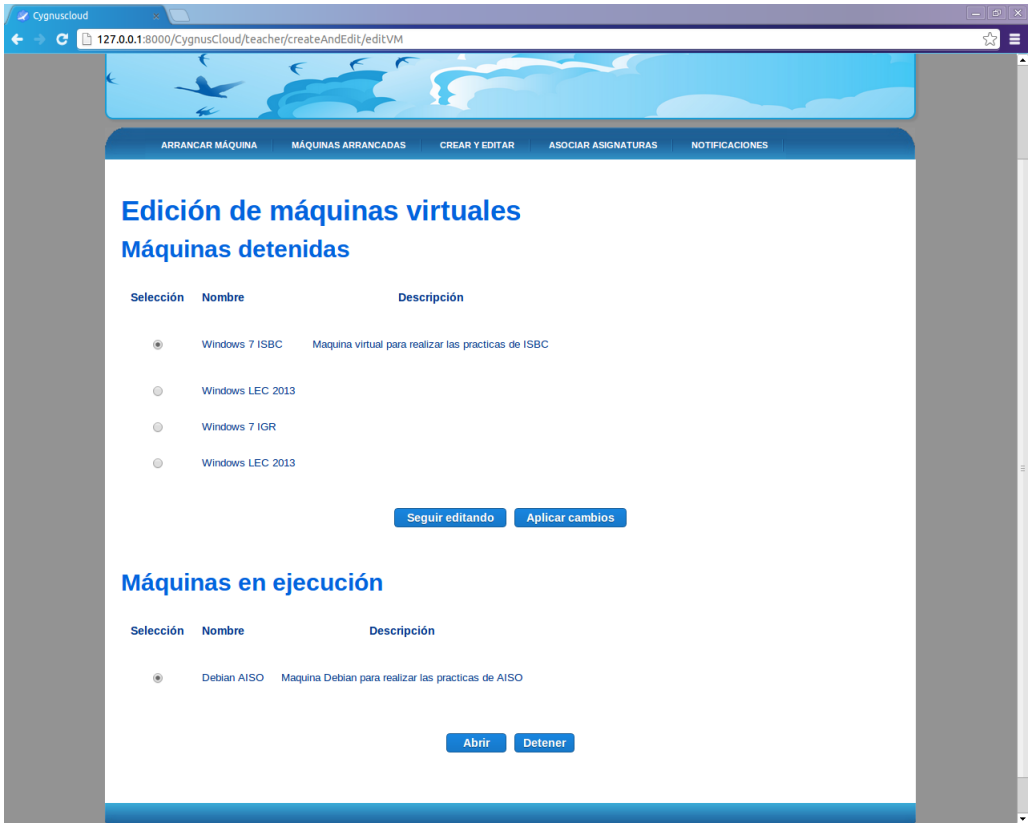

FIGURA 8.23: Página de edición de máquinas virtuales para los profesores

Por último la página de asociación de asignaturas (`Asociar asignatura`) permite al profesor indicar desde cual de las asignaturas que imparte va a ser accesible la imagen que está editando. Esta página esta formada por dos apartados. En el superior el profesor puede seleccionar una de las imágenes en edición y asociarle una asignatura pulsando sobre el botón `Asociar asignatura`.

En el inferior el profesor puede buscar las asignaturas ya asociadas a alguna de las imágenes en edición y desvincularla.

La figura 8.24 muestra una captura de esta página.

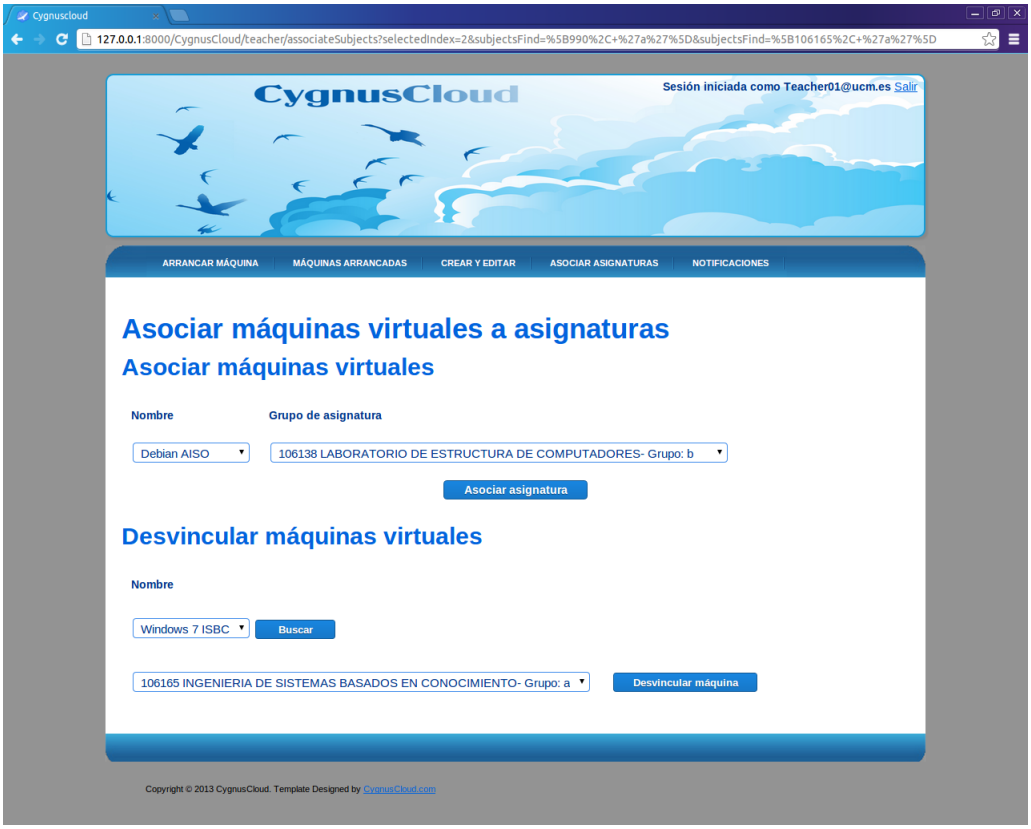

FIGURA 8.24: Página de asociación de asignaturas a máquinas por el profesor

8.4.2.4. Páginas accesibles para los administradores

Al iniciar sesión como administrador, la primera página que se muestra es la página de arranque de máquinas virtuales (`Máquinas virtuales ▷ Arrancar`). La forma de proceder en esta página, al igual que en la página de detención y apertura de máquinas virtuales en ejecución (`Máquinas virtuales ▷ Detener` y `Máquinas virtuales ▷ Abrir`) resulta similar que en el caso de las páginas de estudiantes con una diferencia, como un administrador puede gestionar cualquier máquina virtual existente en el sistema, independientemente de quien la haya creado, es necesario añadir un cuadro de búsqueda en la parte superior de la página que permita filtrar el número de máquinas que se muestran.

La página de edición de máquinas (`Máquinas virtuales ▷` Editar) permite al administrador tomar ciertas decisiones acerca de las máquinas virtuales creadas en el sistema. Así, desde esta página, un administrador puede:

- Eliminar una imagen de toda la infraestructura
- Eliminar una imagen de un determinado servidor de máquinas virtuales
- Desplegar automáticamente un imagen. Con esta opción, será el sistema el que decida en que servidores de máquinas virtuales se desplegará la máquina.
- Desplegar una imagen en un determinado servidor de máquinas virtuales.

La figura 8.25 muestra la página de edición de máquinas virtuales. Al igual que en las páginas anteriores, en la parte superior, podemos observar un cuadro de búsqueda que nos permita filtrar el número de máquinas sobre las que buscar la máquina que va a ser editada.

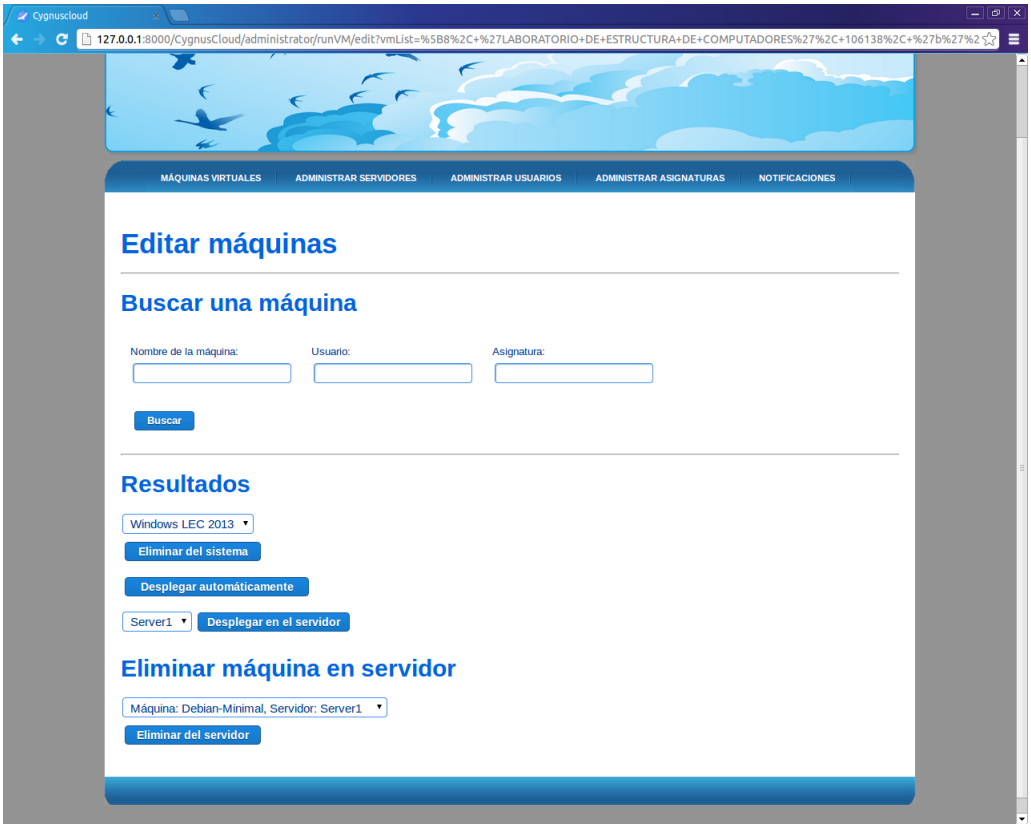

FIGURA 8.25: Página de edición de máquinas virtuales para el administrador

Con respecto a los servidores, los administradores pueden crear y editar servidores por medio de las páginas correspondientes (`Administrar servidores ▷ Añadir servidor` y `Administrar servidores ▷ Editar servidor` respectivamente).

Para crear un nuevo servidor, el administrador deberá indicar el nombre de dicho servidor, único en toda la infraestructura, la dirección IP y el puerto por el cual se conectará el servidor de máquinas virtuales. También, debe indicar, si el servidor será capaz de albergar imágenes base o no. Una vez indicado todo esto, la petición de creación del servidor se enviará pulsando sobre el botón `Añadir servidor`.

En cuanto a la página de edición de servidores, el administrador puede seleccionar el servidor cuyo contenido quiere modificar. Una vez seleccionado, debe pulsar sobre el botón `Buscar servidor` para que se le muestre la información asociada a dicho servidor. Según el estado en el que se encuentre el servidor seleccionado, el administrador podrá aplicar unas operaciones u otras.

En la figura 8.26 se muestra la página de edición de servidores. En ella, ya se ha buscado la información sobre el servidor con nombre `Server1` y se le ha ofrecido al administrador ciertas opciones. En este caso, el administrador podría:

- Cambiar los datos sobre el nombre, puerto, IP e imagen base del servidor.
- Eliminar el servidor de la infraestructura
- Arrancar el servidor

Los administradores también pueden consultar el estado en el que se encuentra los distintos servidores y el repositorio por medio de la página correspondiente (`Administrar servidores ▷ Estado del sistema`). Entendemos como estado, al consumo de recursos de cada uno de los servidores y del repositorio en un momento dado.

Para consultar el estado de un servidor concreto deberá seleccionarse dicho servidor en la lista y pulsar sobre el botón `Mostrar estado`.

La figura 8.27 muestra el estado del servidor Server1 y del repositorio sin ninguna máquina virtual arrancada.

La última página de gestión de servidores permite detener por completo la infraestructura, apagando todos los servidores del sistema (`Administrar servidores ▷ Parar infraestructura`). Esta página esta formada únicamente por una casilla que permite indicar si debe esperarse a que se apaguen todas las máquinas en ejecución para apagar por completo la infraestructura y un botón que permite enviar la petición de apagado.

La figura 8.28 muestra la página de detención de la infraestructura.

El administrador también puede gestionar los usuarios dados de alta en la web. Así, la aplicación web dispone de páginas para la creación y eliminación de usuarios (`Administrar usuarios ▷ Añadir` y `Administrar usuarios ▷ Eliminar` respectivamente). Para añadir un nuevo usuario, el administrador deberá indicar el correo electrónico con el cual accederá el usuario al sistema, la contraseña asociada y el tipo de usuario (estudiante, profesor o administrador). Opcionalmente, puede adjuntar una fotografía del usuario.

En cuanto a la página de eliminación consta de un cuadro superior de búsqueda que permite filtrar el grupo de usuarios por su nombre o tipo de usuario. Una vez realizada la búsqueda, se mostrará una lista de usuarios resultante. Para eliminar el usuario deseado, simplemente se debe seleccionar dicho usuario en la lista y pulsar sobre el botón `Eliminar seleccionado`. La figura 8.29 muestra una captura de la página de eliminación de usuarios.

Una vez creado un usuario, el administrador puede matricularlo en algún grupo de asignatura por medio de la página de asociación de asignaturas (`Administrar usuario ▷ Asociar asignatura`). Al igual que la mayoría de las páginas del administrador, esta página dispone de un cuadro de búsqueda que permite buscar el usuario que quiere matricularse en un determinado grupo de asignatura. Una vez encontrado el usuario, deberá indicarse el código y el grupo de clase del grupo de asignatura en el cual va a ser matriculado el usuario. Todos los grupos de asignaturas tienen asociados un número de plazas disponibles. Antes de asociar un usuario a una determinado grupo, se comprueba que este grupo no se encuentre lleno.

Por último, el administrador también disponen de una par de páginas para crear y eliminar grupos de asignaturas (`Administrar asignatura ▷ Añadir` y `Administrar asignatura ▷ Eliminar` respectivamente). Para crear un nuevo grupo de asignatura, el administrador deberá indicar el nombre de la asignatura que se impartirá en dicho grupo, el código único asociado a dicha asignatura, el año lectivo en el que va a ser cursado dicho grupo, el número de plazas disponibles y el grupo de clase asociado.

Para eliminar un grupo de asignatura existente se procederá de forma análoga a como se indico en la eliminación de usuarios.

FIGURA 8.26: Página de edición de servidores para los administradores

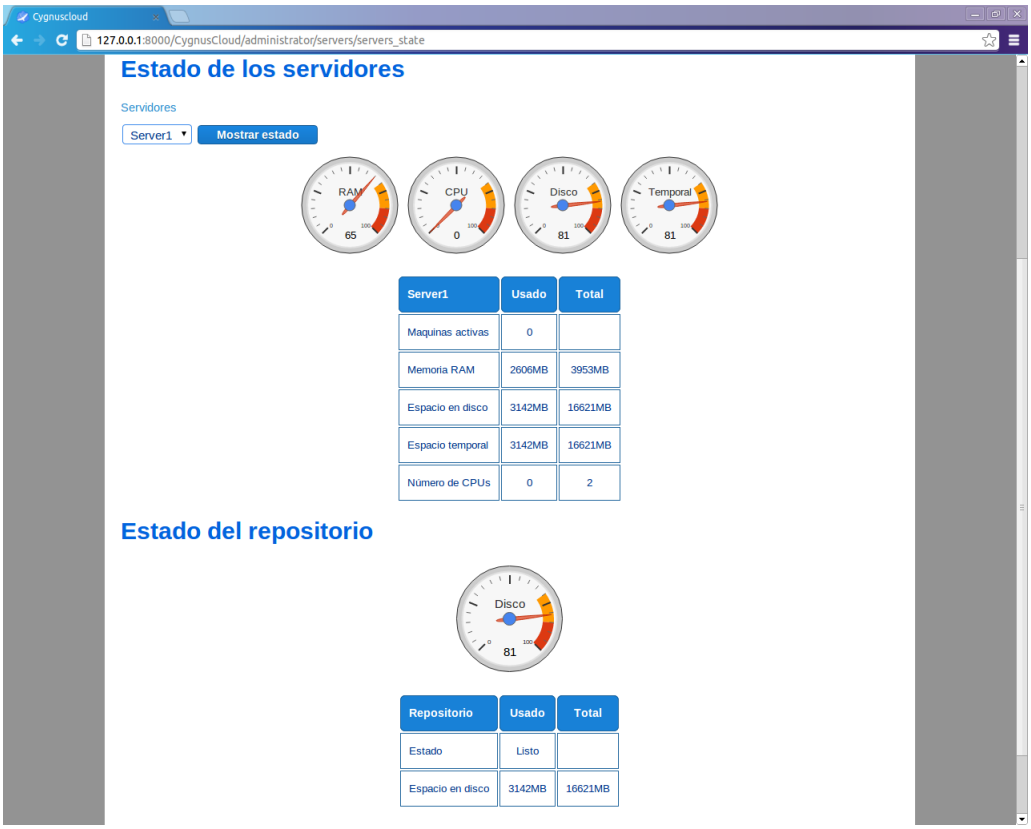

FIGURA 8.27: Página que muestra el estado del sistema

FIGURA 8.28: Página de detención de la infraestructura por el administrador

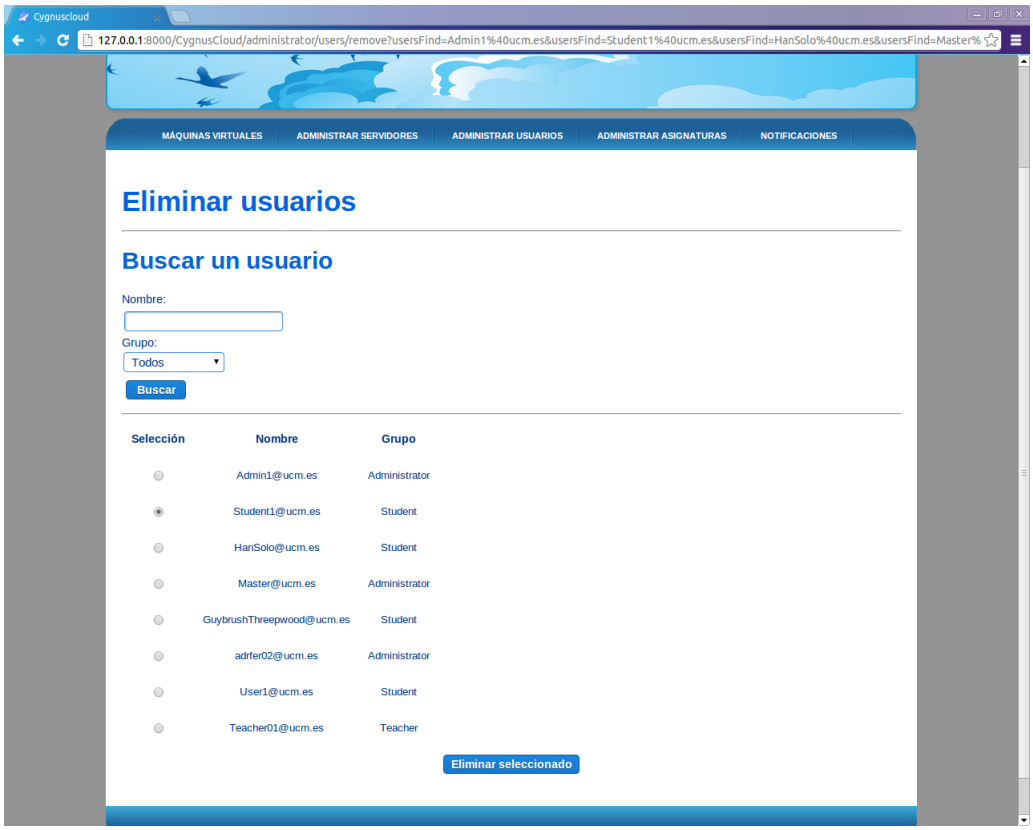

FIGURA 8.29: Página de eliminación de usuarios por parte de un administrador

8.4.2.5. Gestión de notificaciones

Como algunas de las operaciones que ofrece *CygnusCloud* deben ser gestionadas por medio de peticiones, ya que su realización no es inmediata y no podemos tener al usuario esperando mientras

se completa dicha operación, es necesario ofrecer a los usuarios un sistema de notificaciones en el cual puedan consultar los mensajes que le envíe el sistema sobre la finalización errónea o no de las peticiones enviadas.

Cuando el usuario reciba una notificación, aparecerá un etiqueta en la parte superior derecha de la página similar a la que se muestra en la figura 8.30. Esta etiqueta permanecerá mientras el usuario no lea las notificaciones pendientes en la página de notificaciones.

Para acceder a la página de notificaciones, el usuario puede pulsar sobre la etiqueta o bien acceder desde la barra de menú (sección `Notificaciones`).

Una vez el usuario a consultado sus notificaciones pendientes, estas desaparecerán de la página para dejar paso a las nuevas notificaciones que puedan recibirse.

La figura 8.31 muestra un ejemplo de página con notificaciones.

FIGURA 8.30: Etiqueta de notificaciones pendientes en la web

FIGURA 8.31: Página de visualización de notificaciones

8.4.3. Despliegue de la aplicación web

Aunque `web2py` ofrece su propio servidor web para mantener las aplicaciones que se hayan desarrollado, este servidor está más orientado a que los usuarios puedan realizar en él pruebas de depuración de las aplicaciones.

Si el usuario quiere desplegar su aplicación para que pueda ser usado por los clientes tendrá que buscar algún otro tipo de servidor destinado a este fin.

`web2py` ofrece la posibilidad de migrar sus aplicaciones web para ser utilizadas bajo un servidor `Apache`. Con este fin, `web2py` ofrece un par de *scripts* de configuración, bajo el directorio `/web2py/scripts`, que permite instalar todos los paquetes necesarios para poder realizar el cambio de servidor.

Podemos encontrar dos *scripts* diferentes:

- `setup-web2py-fedora.sh` (para realizar la instalación en Fedora Core).
- `setup-web2py-ubuntu.sh` (para la instalación en Ubuntu).

Si lo prefiere, el usuario puede realizar la instalación de forma manual siguiendo las especificaciones detalladas en el manual de referencia de `web2py` [15].

Apéndices

Apéndice A

Licencia

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

En el resto de casos, se aplicarán a cada tipo de contenido los términos que aparecen a continuación.

A.1. Código fuente

A.1.1. Versión modificada de noVNC

Copyright © 2012, Joel Martin (https://github.com/kanaka)

Copyright © 2013, Luis Barrios Hernández, Adrián Fernández Hernández, Samuel Guayerbas Martín

This Source Code Form is subject to the terms of the Mozilla Public License, version 2.0. If a copy of the MPL was not distributed with this file, you can obtain one at

http://mozilla.org/MPL/2.0/

A.1.2. Resto de código fuente de *CygnusCloud*

Copyright © 2013, Luis Barrios Hernández, Adrián Fernández Hernández, Samuel Guayerbas Martín

The source code is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

A.2. Documentación

Toda la documentación del proyecto se distribuye bajo la siguiente licencia siempre que no se especifique lo contrario:

Apéndice B

Git: guía de referencia

Este capítulo contiene material extraído del libro *Pro Git*, escrito por Scott Chacon y publicado bajo una licencia *Creative Commons* Attribution-NonCommercial-ShareAlike 3.0 Unported.

B.1. Git y Subversion

Generalmente, los repositorios más utilizados, tales como SVN, se encuentran estructurados como se indica en la figura B.1. Existe un único repositorio central, al que se van haciendo *commits* cuando es necesario. En estos casos, lo más normal es trabajar en un directorio separado, pegar los ficheros modificados en el directorio donde está la copia del repositorio y hacer el *commit*.

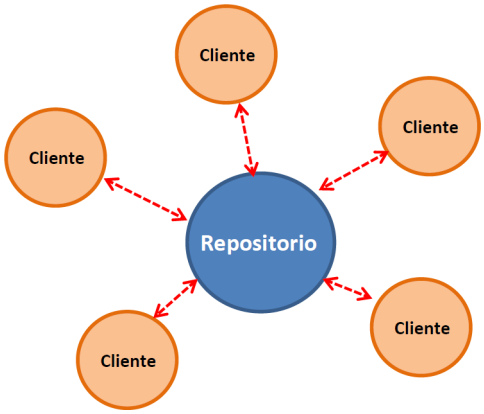

FIGURA B.1: Subversion

Con Git la situación es bastante diferente, siguiendo la estructura representada en la figura B.2. Git es un sistema de control de versiones distribuido, por lo que no existe el gran repositorio central: todos los repositorios tienen la misma importancia.

La forma de trabajar en Git es bastante distinta: cada desarrollador tendrá su repositorio local, al que irá haciendo *commits* cuando quiera. Es posible descargar información de los repositorios de otros (repositorios remotos), y también es posible subir información a esos repositorios. Así, el repositorio de Google Code será un repositorio más.

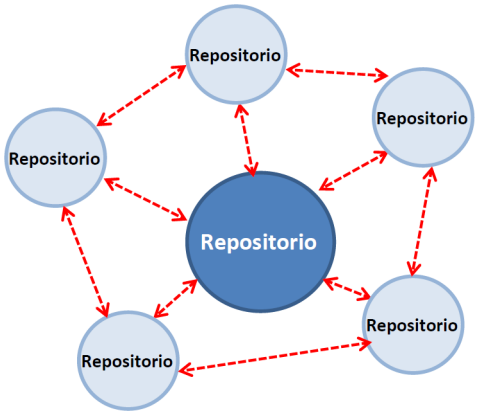

FIGURA B.2: Git

B.2. Instalación y configuración básica de Git

Para empezar a utilizar Git, es necesario disponer del binario instalado en nuestro sistema. Para ello, abrimos un terminal y escribimos

```
$ sudo apt-get install git gitk
```

Una vez hecho esto, fijaremos algunas cosas importantes (como el nombre de usuario) y otras que no lo son tanto, pero que facilitan el uso de Git.

Para determinar el editor que se utilizará a la hora de escribir los commit, escribimos

```
export EDITOR=nano
```

al final del fichero `~/.bashrc`.

Lo siguiente que haremos es introducir nuestro nombre de usuario y dirección de correo electrónico. Esto no hace falta para conectarse a Google Code: simplemente se utilizará para firmar los *commits*. Para ello, abrimos un terminal y escribimos

```
$ git config --global user.name "<Nombre de usuario>"
$ git config --global user.email "<Correo electrónico>"
```

El *flag* `--global` indica que utilizaremos estos ajustes en *todos* los repositorios. Si queremos utilizarlos solamente en algunos, iremos al directorio del repositorio y ejecutaremos estas órdenes sin el *flag* `--global`.

Finalmente, cambiaremos un poco los colores predeterminados para facilitar la lectura de los mensajes de Git. Para ello, añadimos este texto al fichero `~/.gitconfig`.

```
[color]
     ui = true
[color "branch"]
     current = yellow reverse
     local = yellow
     remote = green
[color "diff"]
     meta = yellow bold
     frag = magenta bold
     old = red bold
     new = green bold
     whitespace = red reverse
[color "status"]
     added = yellow
     changed = green
     untracked = cyan
[core]
     whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol
```

B.3. Uso básico de Git

B.3.1. Registrar cambios en el repositorio

Ahora que tenemos un repositorio, ha llegado el momento de utilizarlo. Cada uno de los ficheros del directorio de trabajo puede estar en dos estados:

- *tracked* (registrados). Se trata de ficheros que ya estaban en el repositorio. A su vez, pueden estar en tres estados: *unmodified* (actualizados en el repositorio), *modified* (sin actualizar en el repositorio) o *staged* (ficheros modificados *que hay que subir* al repositorio).
- *untracked* (no registrados). En este grupo están el resto de ficheros.

El ciclo de vida de un fichero en Git aparece en la figura B.3.

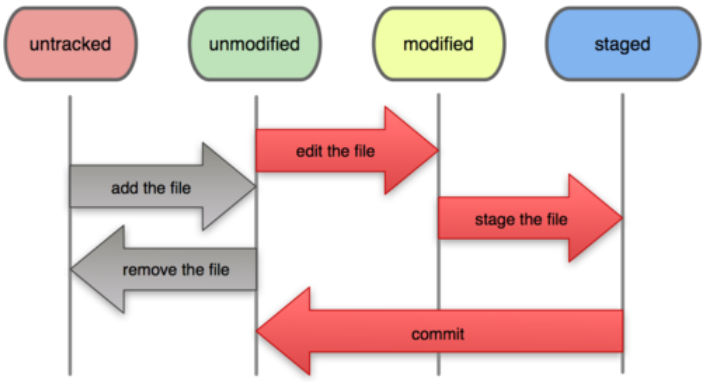

FIGURA B.3: Ciclo de vida de un fichero en Git

Cuando se clona el repositorio, todas los ficheros estarán en estado *tracked* y *unmodified* (aún no se ha hecho nada con ellos). A medida que se editan, pasan a estar en estado *modified*. Finalmente, pasan a estado *staged* y se hace un *commit* para reflejar los cambios en el repositorio.

Todos los ficheros que se subirán al repositorio al hacer un *commit* se encuentran en la *staging area*.

B.3.1.1. Comprobando el estado de los ficheros

Para determinar el estado de los ficheros se utiliza la orden `git status`. Justo después de clonar el repositorio, su salida es

```
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
nothing to commit (working directory clean)
```

Como era de esperar, no ha habido cambios con respecto al repositorio. Es más, no existe ningún fichero nuevo, ya que en ese caso habría aparecido en la salida. Además, el comando nos indica que estamos utilizando la rama (*branch*) `master`, que es la rama por defecto. Si añadimos algo, como el fichero `README.txt`, la salida del comando `git status` ya es diferente.

```
luis@luis-laptop:~/cygnus-cloud$ touch README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Untracked files:
#    (use "git add <file>..." to include in what will be committed)
#
#    README.txt
nothing added to commit but untracked files present (use "git add" to track)
```

B.3.1.2. Incluyendo nuevos ficheros

Para evitar errores, Git *no* incluirá ficheros en estado *untracked* a no ser que se le pida explícitamente. Para ello, se utiliza la orden `git add`.

```
luis@luis-laptop:~/cygnus-cloud$ git add README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Changes to be committed:
#    (use "git reset HEAD <file>..." to unstage)
#
# new file: README.txt
#
```

Al ejecutar nuevamente la orden git status, podemos comprobar que el fichero `README.txt` está en estado *tracked* y *staged*.

En general, la sintaxis de la orden `git add` es

```
git add <ruta>
```

Si `<ruta>` es un directorio, se añaden recursivamente todos los ficheros. Si es un fichero, se añade únicamente ese fichero.

B.3.1.3. Modificando ficheros ya existentes

Supongamos que, tras hacer el *commit*, modificamos el fichero `README.txt`. Tras ejecutar la orden git status, obtendríamos algo como esto:

```
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Changes not staged for commit:
#    (use "git add <file>..." to update what will be committed)
#    (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: README.txt
#
```

¿Qué ocurre? El fichero `README.txt` ha sido modificado. Para indicar que queremos que los cambios sean permanentes, hay que conseguir que el fichero pase a estado *staged*. Para ello volvemos a utilizar la orden `git add`. Ahora, la salida será

```
luis@luis-laptop:~/cygnus-cloud$ git add README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Changes to be committed:
#    (use "git reset HEAD <file>..." to unstage)
#
# modified: README.txt
#
```

Supongamos que introducimos una modificación de última hora. Si volvemos a ejecutar la orden `git status`, obtendremos algo como

```
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Changes not staged for commit:
#    (use "git add <file>..." to update what will be committed)
#    (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: README.txt
#
```

Hemos vuelto al mismo estado que al principio ya que los datos que se añadirán al repositorio al hacer el *commit* se fijan en el momento en que se ejecuta la orden `git add`. Si tras ejecutar esta orden se realiza otra modificación, será necesario volver a ejecutar el comando `git add` antes de hacer el *commit* si queremos que los nuevos cambios se reflejen en el repositorio.

B.3.1.4. Ignorando ficheros y directorios

Pueden existir ficheros irrelevantes que no queremos que estén en el repositorio. Es el caso de los ejecutables, ficheros temporales, *logs*, ficheros auxiliares...

Para excluirlos, basta con crear el fichero `.gitignore` en el directorio de trabajo y escribir en él unas cuantas expresiones regulares. El contenido de uno de estos ficheros podría ser este:

```
luis@luis-laptop:~/cygnus-cloud$ cat .gitignore
*~
*.class
*.[oa]
```

Con esto, estamos ignorando los ficheros temporales, los ficheros `.class` (*bytecode* de la JVM) y los ficheros de código objeto (con extensión `.o` o `.a`).

Las reglas para generar estos ficheros son las siguientes:

- Las líneas en blanco y las que empiezan por almohadilla (`#`) son ignoradas.
- Las expresiones regulares deben ser patrones glob. Podemos encontrar más información sobre esto en aquí.
- Los patrones que terminan por `/` hacen referencia a directorios.
- Los patrones que comienzan por admiración (`!`) están negados.

B.3.1.5. Nuestro primer *commit*

Una vez que tenemos claro qué queremos subir al repositorio, ha llegado el momento de hacer nuestro primer *commit*. Es importante notar que todo lo que esté en estado *unstaged* (cambios no confirmados) *no* se subirá al repositorio.

Para hacer el *commit*, basta con ejecutar la orden `git commit`. Al hacerlo, se abrirá el editor que fijamos en la sección B.2. Bastará con escribir un mensaje, guardar el fichero y cerrar el editor. La salida de la orden será algo como

```
luis@luis-laptop:~/cygnus-cloud$ git commit
[master c6733d0] + Subo Readme.txt
1 file changed, 1 insertion(+), 1 deletion(-)
```

Esta salida nos está indicando que hemos añadido nuestro fichero a la rama `master`, la suma SHA-1 del *commit*, el número de ficheros modificados y también el número de líneas de código añadidas y eliminadas.

También es posible hacer un *commit* usando el flag `-m "Mensaje"`.

B.3.1.6. *Commits* "rápidos"

Aunque es muy útil saber exactamente qué vamos a subir al repositorio, a veces lo único que hace falta es subir todo lo que haya en el directorio de trabajo y que no deba ignorarse.

Es posible saltarnos el uso de `git add` y hacer directamente el *commit* con todo proporcionándole el flag `-a` al comando `git commit`.

B.3.1.7. Borrando ficheros del repositorio

Para eliminar un fichero del repositorio, basta con utilizar la orden `git rm`. Esta orden también eliminará el fichero del directorio de trabajo.

Por seguridad, si el fichero vuelve a crearse y a registrarse, no se borrará al hacer un *commit*. Para forzar el borrado en estos casos, es necesario proporcionar el flag `-f` a la orden `git rm`.

Por otra parte, resulta interesante eliminar un fichero del repositorio y mantenerlo en el directorio de trabajo. Para conseguir esto, basta con utilizar la orden

```
$ git rm --cached <fichero>
```

La orden `git rm` puede recibir como argumento rutas de ficheros, directorios y patrones glob que hagan referencia a ficheros. En caso de utilizar patrones glob, es importante notar que todos los asteriscos deben ir precedidos por \ (para evitar la expansión de nombres que ya utiliza Git).

B.3.1.8. Moviendo y renombrando ficheros

En principio, si se renombra uno de los ficheros del repositorio, no habrá metadatos que se lo indiquen a Git. Para evitar problemas, es preferible utilizar la orden `git mv`. Su sintaxis es

```
$ git mv <ruta fichero fuente> <ruta fichero destino>
```

Naturalmente, esto también puede hacerse a mano. La orden anterior equivale a

```
$ mv <ruta fichero fuente> <ruta fichero destino>
$ git rm <ruta fichero fuente>
$ git add <ruta fichero destino>
```

B.3.2. Revisar cambios en el repositorio

Ahora que hemos hecho nuestros primeros *commits*, podemos consultar como se ha modificado el repositorio. Para ello, utilizaremos la orden `git log`. Al ejecutarla en nuestro repositorio, obtenemos esta salida:

```
luis@luis-laptop:~/cygnus-cloud$ git log
commit bdd9360b437b8a38cf866879f731ae568309aefd
Author:  Luis Barrios <luisbarrioshdez@gmail.com>
Date:    Fri Sep 14 19:49:57 2012 +0200
    Plantillas de la documentación revisadas
```

Como podemos ver, aparecen las *checksums* SHA-1, el nombre y el correo del autor, la fecha y el mensaje de cada *commit*.

Evidentemente, es posible alterar este comportamiento cambiando los argumentos de la orden. Algunos muy útiles aparecen en el cuadro B.1.

Flag	Descripción de la salida
`-p`	Muestra los cambios introducidos en los *commits* (vía `diff`)
`--stat`	Muestra abreviadamente los cambios introducidos en los *commits*
`-n`	Restringe la salida a los últimos `n` *commits*
`--relative-date`	Usa fechas relativas (i.e. `10 days ago`)
`--since=<fecha>, --after=<fecha>`	Muestra los *commits* realizados desde la fecha `<fecha>`. Las fechas pueden ser específicas (`"2012-09-14"`) o relativas (`2.years.1.week.10.hours`)
`--until=<fecha>, --before=<fecha>`	Muestra los *commits* realizados hasta la fecha `<fecha>`. Las fechas pueden ser específicas (`"2012-09-14"`) o relativas (`3.weeks.10.hours`)
`--author`	Muestra los *commits* con la entrada `author` especificada.
`--committer`	Muestra los *commits* con la entrada `committer` especificada.

CUADRO B.1: Argumentos muy útiles de `git log`

B.3.3. Deshacer cambios

Importante: *no* todos los procedimientos que se describen en esta sección son reversibles por lo que es necesario tener mucho cuidado a la hora de realizarlos.

B.3.3.1. Cambiar el último *commit*

Como ya hemos visto, no todos los ficheros que se encuentran en el directorio de trabajo se subirán al repositorio al hacer un *commit*: sólo se subirán aquellos ficheros que se encuentren en estado *staged*. Se dice que todos estos ficheros forman parte de la *staging area*.

Para añadir más ficheros al último *commit* o cambiar el mensaje, se utiliza la orden

```
git commit --amend
```

B.3.3.2. Eliminar un fichero en estado *staged*

Para dejar este apartado lo suficientemente claro, incluiremos un ejemplo. Empezamos añadiendo el fichero `basura.txt` al directorio de trabajo.

```
luis@luis-laptop:~/cygnus-cloud$ touch basura.txt
```

Ahora, añadiremos todos los ficheros a la *staging area*.

```
luis@luis-laptop:~/cygnus-cloud$ git add . luis@luis-laptop:~/cygnus-cloud$
git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#      new file: basura.txt
#
```

Sin embargo, no queremos incluir el fichero `basura.txt` en nuestro *commit*. Para eliminarlo de la *staging area*, la propia orden `git status` nos indica que podemos hacer.

```
luis@luis-laptop:~/cygnus-cloud$ git reset HEAD basura.txt
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#      basura.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

B.3.3.3. Deshacer los cambios en un fichero

Aquí también utilizaremos un ejemplo. Empezaremos escribiendo algo en nuestro fichero `en_blanco.txt`.

```
luis@luis-laptop:~/cygnus-cloud$ echo "foo" > en_blanco.txt
```

Queremos dejar el fichero `en_blanco.txt` vacío, tal y como estaba en nuestro último *commit*. La orden `git status` también nos indica cómo hacerlo.

```
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#      modified: en_blanco.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Ejecutando la orden `git checkout`, la cosa queda así:

```
luis@luis-laptop:~/cygnus-cloud$ git checkout -- en_blanco.txt
luis@luis-laptop:~/cygnus-cloud$ git status
# On branch master
nothing to commit (working directory clean)
```

B.3.4. Uso de repositorios remotos

Un repositorio remoto no es más que un repositorio git alojado en otra máquina. Para mostrar los repositorios remotos que estamos utilizando, es posible utilizar la orden `git remote`.

```
$ git remote
origin
```

El *flag* `-v` muestra también la URL de cada repositorio remoto.

```
$ git remote -v
origin https://code.google.com/p/cygnus-cloud/ (fetch)
origin https://code.google.com/p/cygnus-cloud/ (push)
```

En principio, no es necesario añadir repositorios remotos para empezar a trabajar. Para hacerlo, basta con utilizar la orden

```
git remote add [nombre-del-repositorio] [url]
```

B.3.4.1. Incorporar los cambios de un repositorio remoto al repositorio local

Para añadir al directorio de trabajo todos los cambios que se han producido en un repositorio remoto, basta con utilizar la orden

```
git fetch [nombre-del-repositorio]
```

Cuando se clona un repositorio, el repositorio clonado pasa a llamarse `origin`. Como hemos clonado el repositorio de Google Code, `git fetch origin` añadirá todos los cambios del repositorio de Google Code al repositorio local.

Importante: la orden `git fetch` *no* mezcla el contenido del repositorio local y del repositorio remoto. Esa mezcla *debe* hacerse a mano.

El uso directo de `git fetch` resulta bastante engorroso. La situación más habitual es esta:

- en el repositorio remoto existen varias ramas o *branches*, y
- nos interesa el contenido de *una* de esas ramas. Por ejemplo, una rama con código experimental no es interesante, mientras que la rama `master`, que contiene el código estable, sí lo es.

En estos casos, lo que se hace es utilizar la orden

```
git pull [nombre-del-repositorio] [nombre-de-la-rama]
```

Esta orden toma los datos de la rama especificada y trata de mezclarlos con lo que hay en el directorio de trabajo. Si hay conflictos, tendremos que resolverlos a mano.

B.3.4.2. Incorporar los cambios del repositorio local a un repositorio remoto

En esta sección aprenderemos a subir cosas al repositorio en Google Code. El comando que hay que utilizar es

```
git push [nombre-del-repositorio] [nombre-de-la-rama]
```

Por ejemplo, si queremos subir los cambios en la rama master a Google Code, basta con ejecutar `git push origin master`.
Solamente es posible subir cosas a un repositorio remoto cuando se dispone de la última versión de sus datos. Esto ya ocurre en otros sistemas de control de versiones, como SVN.

B.3.4.3. Renombrar y eliminar repositorios remotos

Para cambiar el nombre de un repositorio remoto, basta con utilizar la orden

```
git remote rename [nombre-del-repositorio] [nuevo-nombre-del-repositorio]
```

Para eliminar un repositorio remoto, basta con utilizar la orden

```
git remote rm [nombre-del-repositorio]
```

B.3.5. Uso de etiquetas

Las etiquetas son sólo formas de marcar puntos importantes de la historia. Lo más habitual es que se correspondan con versiones.

B.3.5.1. Crear etiquetas

En Git existen dos clases de etiquetas: ligeras (*lightweight*) y anotadas (*annotated*). Las primeras no son más que el *checksum* de un *commit*, mientras que las segundas almacenan más información, como un mensaje y una fecha.

Para crear una etiqueta anotada, basta con utilizar la orden `git tag -a`.

```
$ git tag -a PlantillasV3 -m 'Plantillas de la documentación'
$ git tag
PlantillasV3
```

Para visualizar el contenido asociado a una etiqueta anotada, se utiliza la orden `git show`.

```
$ git show PlantillasV3
tag PlantillasV3
Tagger: Luis Barrios <luisbarrioshdez@gmail.com>
Date:   Mon Sep 24 18:17:22 2012 +0200
Plantillas de la documentación
```

Para crear una etiqueta ligera, basta con no proporcionar ningún *flag* (es decir, sólo el nombre de la etiqueta) a la orden `git tag`.

Con lo que hemos visto hasta ahora, sólo es posible asociar etiquetas al último *commit*. Para asociar las etiquetas a un *commit* anterior, basta con proporcionar el *checksum* o una parte del *checksum* de ese *commit* como último argumento de la orden `git tag`.

B.3.5.2. Compartir etiquetas

De forma predeterminada, el comando git push *no* transfiere las etiquetas a los repositorios remotos, por lo que habrá que hacerlo de forma explícita. Para ello, basta con utilizar la orden

```
git push [nombre-del-repositorio] [nombre-de-la-etiqueta]
```

Para transferir todas las etiquetas definidas al repositorio remoto, es posible utilizar la orden

```
git push --tags
```

para no hacerlo etiqueta a etiqueta.

B.4. Uso de ramas o *branches*

B.4.1. ¿Qué es una rama?

Para entender exactamente qué es una rama o *branch* y cómo se utilizan en Git hay que tener en cuenta cómo se almacenan los datos.

En la mayoría de sistemas de control de versiones, los datos se almacenan como un conjunto de ficheros y de cambios realizados sobre los mismos a lo largo del tiempo. Esto se muestra en la figura B.4.

En cambio, Git almacena los datos como un conjunto de instantáneas de un mismo "sistema de ficheros". Cada vez que se realiza un *commit*, Git almacena el "estado" de esos ficheros en una instantánea junto con un puntero a la misma. Por eficiencia, si un fichero no ha cambiado, solamente se almacena un enlace a la versión que ya estaba almacenada. Esto aparece en la figura B.5.

Al hacer un *commit* no solamente se almacena un puntero a la instantánea: también se almacenan metadatos (como el autor y el mensaje) y uno o más punteros a los *commits* padre. Así,

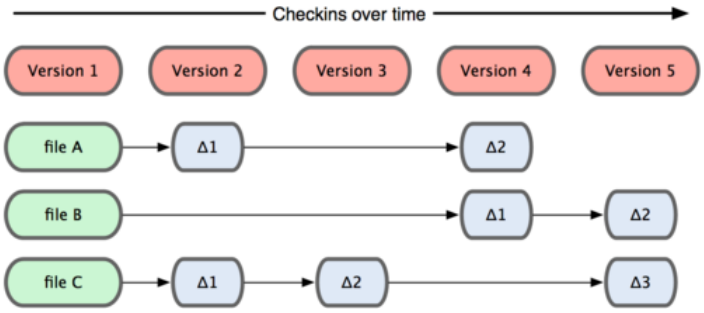

FIGURA B.4: Almacenamiento de los datos en la mayoría de sistemas de control de versiones (CVS, SVN, Bazaar,...)

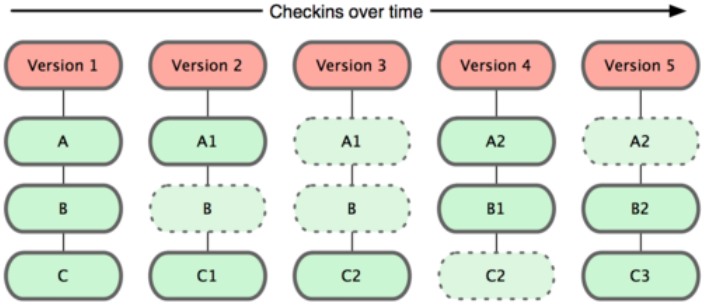

FIGURA B.5: Almacenamiento de los datos en Git

- el primer *commit* no tendrá ningún puntero a su padre.
- los *commits* "normales" tienen un puntero a su padre.
- los *commits* que resultan de mezclar dos o más ramas tienen varios punteros.

Con el fin de asentar estas ideas, utilizaremos un ejemplo. Supongamos que ejecutamos las órdenes

```
$ git add README test.rb LICENSE
$ git commit -m 'initial commit of my project'
```

Tras el *commit*, los datos del repositorio se organizan de acuerdo a la figura B.6. Los *blobs* no son más que las versiones de los distintos ficheros.

Si introducimos cambios y hacemos otro *commit*, este almacenará un puntero a su *commit* padre. Tras dos *commits* más, la historia sería como la de la figura B.7.

Una rama es un puntero a un *commit*. La rama por defecto, que ha aparecido ya en todos nuestros *commits*, se llama `master`.

Cada vez que realizamos un *commit*, se parte de una rama principal que apunta a nuestro último *commit*. Tras cada *commit*, este puntero avanzará automáticamente. Esto se muestra en la figura B.8.

B.4.2. Uso básico de ramas

Para crear una rama, se utiliza el comando `git branch`. Si ejecutamos el comando

```
$ git branch testing
```

Se creará un nuevo puntero al último *commit*. Esto aparece en la figura B.9.

Git averigua la rama que hay que utilizar manteniendo un puntero especial, `HEAD`, que apunta a la rama *local* en la que se está trabajando.

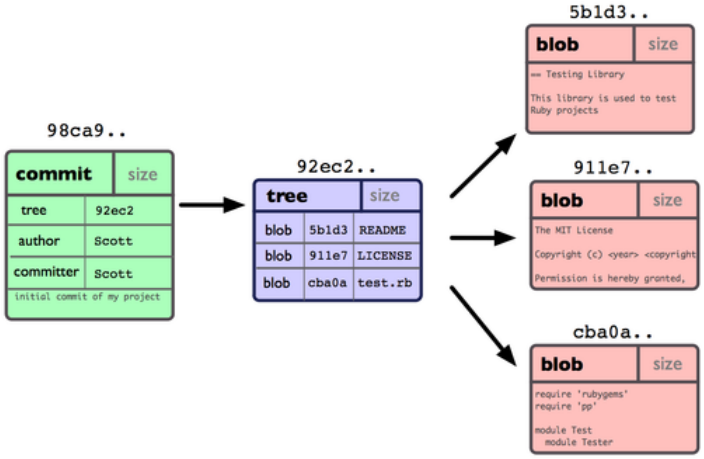

FIGURA B.6: Organización de los datos tras un *commit*

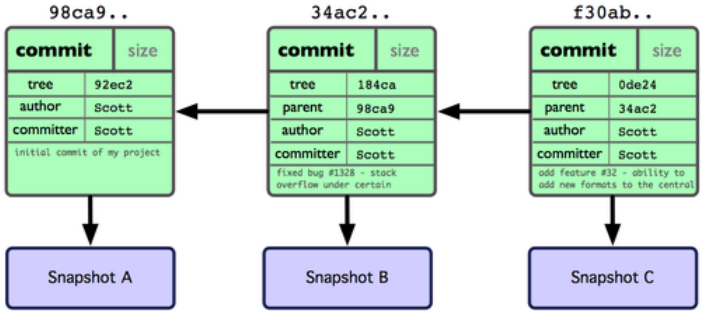

FIGURA B.7: Historia tras dos *commits* más

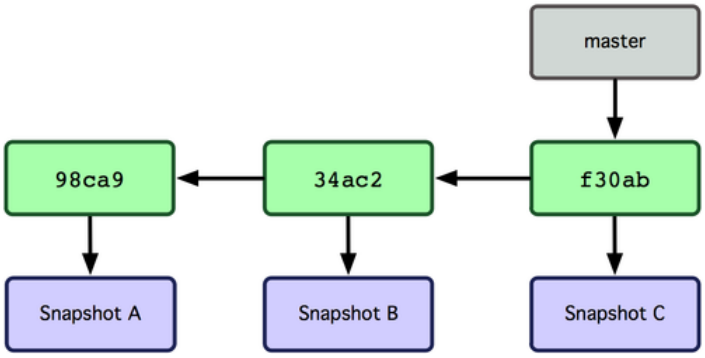

FIGURA B.8: *Branches* y *commits*

El comando `git branch` *solamente* crea una rama, pero *no* cambia de rama. Para utilizar una rama ya existente, se utiliza el comando `git checkout`. Si ejecutamos la orden

```
$ git checkout testing
```

la situación será la de la figura B.10.

Si hacemos otro *commit*,

```
$ echo "foo" >> LICENSE
$ git commit -a -m 'Cambio en LICENSE'
```

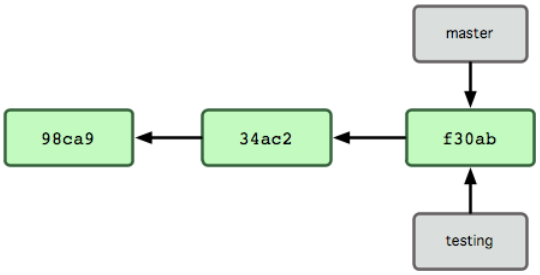

FIGURA B.9: Creación de una nueva rama

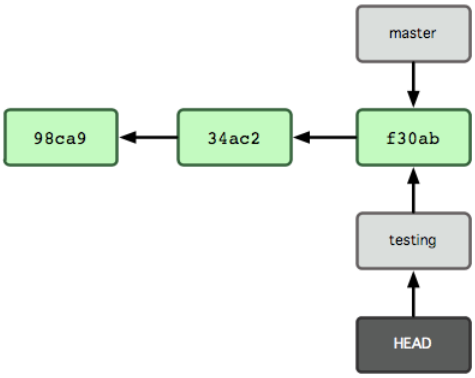

FIGURA B.10: Situación tras ejecutar `git checkout testing`

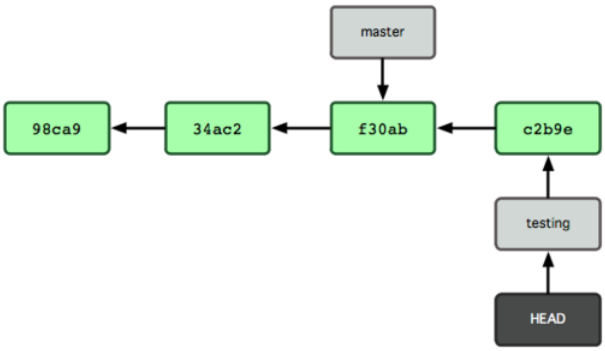

FIGURA B.11: *Commit* en la rama *testing*

la situación pasa a ser la de la figura B.11.

Mientras que la rama `testing` ha avanzado, `master` sigue apuntando al mismo *commit*. Si ahora ejecutamos

```
$ git checkout master
```

se habrán deshecho todos los cambios. Es decir, cada rama tiene asociada su propia historia.

Importante: para evitar problemas, es muy recomendable cambiar de rama *después* de realizar un *commit*.

Las ramas son muy útiles para hacer cambios de forma controlada, pero tarde o temprano esos cambios deberán incorporarse a la rama `master`. Para incorporar los cambios de una rama a otra se utiliza la orden `git merge`. Así, los comandos

```
$ git checkout master
$ git merge testing
```

incorporan todos los cambios de la rama testing a la rama master. Dado que el último *commit* de master no tiene por qué ser el padre del último *commit* de testing[1], Git deberá trabajar algo más. En cualquier caso, los conflictos que aparezcan deberán resolverse a mano.

Finalmente, las ramas se borran proporcionando el flag `-d` a la orden `git branch`. Así, el comando

```
$ git branch -d testing
```

eliminará la rama `testing`.

B.4.2.1. Resolución de conflictos

En algunos casos, mezclar ramas no es trivial. Si se ha modificado el mismo fragmento del mismo fichero en las dos ramas que se están mezclando, aparecerá un conflicto. Por ejemplo, si tenemos dos ficheros `foo.xml` en la raíz del directorio, el último *commit* de `master` no es el ancestro del de `testing` y su contenido es

```
<foo>
    foo
</foo>
```

y

```
<foo>
    moo
</foo>
```

Esto producirá un conflicto.

Cuando aparece un conflicto, no se crea el *commit* de la mezcla: la mezcla se pausa hasta que se resuelve manualmente el conflicto.

Para resolver conflictos:

1. Se modifican los fragmentos correspondientes en un editor (escogiendo una alternativa y quitando las líneas `<<<<<<<`, `=======` y `>>>>>>>`).
2. Se ejecuta el comando `git add` sobre los dos ficheros para marcar el conflicto como resuelto.
3. Se realiza un *commit*.

B.4.3. Gestión de ramas

El comando `git branch` hace algo más que crear y borrar ramas. Si se ejecuta sin argumentos, lista las ramas existentes. El nombre de la rama en la que se está trabajando va precedido por el carácter `*`.

```
$ git branch
* master
  testing
```

Para determinar cuál es el último *commit* realizado en cada rama, es posible utilizar el comando `git branch -v`.

```
$ git branch -v
* master  20aaf1c Otro commit más
  testing 81f2925 Commit de prueba
```

También es útil saber qué ramas se han mezclado con la actual y cuáles no. Para ello, se utilizan los comandos `git branch --merged` y `git branch --no-merged`.

[1] Cuando esto sucede, Git se limita a mover el puntero de la rama a la que se incorporan los cambios (`Fast forward`)

```
$ git branch --merged
* master
  testing
$ git branch --no-merged
  foo
```

B.4.4. Ramas remotas

Las ramas remotas son ramas locales que no se pueden modificar. Las modificaciones en estas ramas se realizan intercambiando datos a través de redes, y sus nombres son de la forma

```
(repositorio remoto)/(rama)
```

Así, la rama `master` del repositorio de Google Code es `origin/master`.

B.4.4.1. Acceso a ramas remotas

Para incorporar los cambios en las ramas remotas al directorio de trabajo, basta con ejecutar la orden

```
$ git fetch origin
```

Con esto, descargaremos todos los cambios en el repositorio de Google Code a nuestro directorio de trabajo.

B.4.4.2. Crear ramas remotas

Para compartir una rama, hay que registrarlas en el repositorio remoto de Google Code. Las ramas locales no se sincronizan automáticamente con las remotas, por lo que hay que registrarlas manualmente.

Para crear una rama remota, basta con ejecutar la orden

```
$ git push (repositorio remoto) (rama)
```

Es importante notar que, tras leer los datos de una rama remota, no se dispone de una copia local editable. Por ejemplo, tras ejecutar las órdenes

```
$ git push origin foo
$ git fetch origin
```

no existirá la nueva rama `foo`: sólo existirá el puntero `origin/foo`, que *no* podrá modificarse. La copia editable se obtiene mezclando esta rama con la actual, es decir, ejecutando la orden

```
$ git merge origin/foo
```

B.4.4.3. *Tracking branches*

Las *tracking branches* son ramas locales creadas a partir de una rama remota, es decir, ramas locales que tienen una relación directa con una rama remota. Así, al trabajar con una de estas ramas, Git sabe dónde subir los cambios al ejecutar la orden `git push`, y de dónde obtener datos al ejecutar la orden `git pull`.

Al clonar un repositorio, siempre se crea una *tracking branch*, `master`, relacionada con la rama `origin/master`. Esa es la razón por la que `git push` y `git pull` funcionan sin argumentos.

Naturalmente, estas ramas también pueden crearse a mano, mediante la orden

```
$ git checkout --track (repositorio remoto)/(rama)
```

B.4.4.4. Borrar ramas remotas

Para borrar una rama remota, basta con utilizar la orden

```
$ git push (repositorio remoto) :(rama)
```

B.4.5. Uso de `rebase`

En Git, existen dos formas de integrar los cambios de una rama en otra: `merge` y `rebase`. En esta sección, presentaremos el uso básico de la segunda.

B.4.5.1. Uso básico

Supongamos la situación de la figura B.12, en la que existen dos *commits* en dos ramas diferentes.

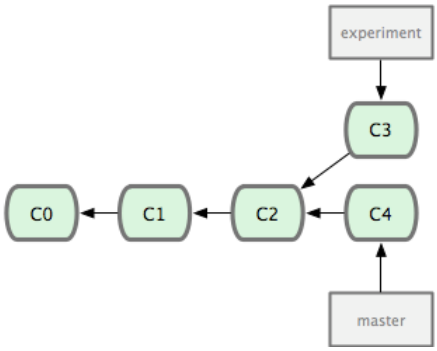

FIGURA B.12: Situación inicial (antes de utilizar `rebase`)

La forma más sencilla de integrar los cambios de las dos ramas es utilizando el comando `merge`. La situación pasaría a ser la de la figura B.13.

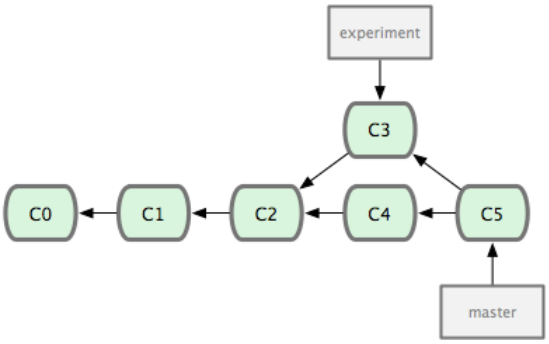

FIGURA B.13: Integrando los cambios con `merge`

Pero existe una alternativa: podemos tomar los cambios del *commit* `C3` y aplicarlos sobre el *commit* `C4`. Esto se llama *rebasing*. Así, el comando

```
$ git rebase master
```

tomará todos los cambios introducidos en los *commits* de `experiment` y los aplicará sobre `master`. La situación pasará a ser la de la figura B.14.

Si bien el resultado final es el mismo, al usar `rebase` en este caso la historia pasa a ser lineal y los *commits* cambian.

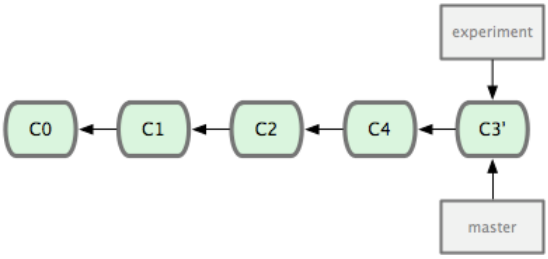

FIGURA B.14: Integrando los cambios con `rebase`

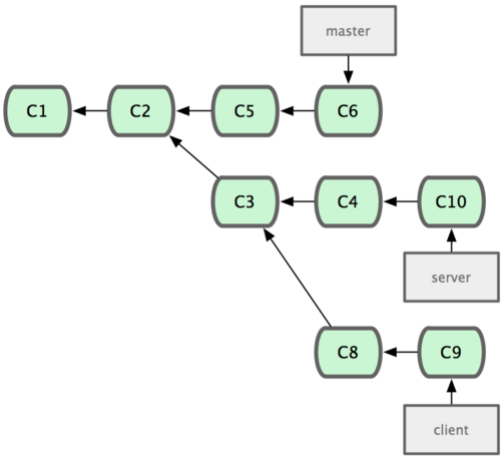

FIGURA B.15: Uso útil de `rebase`. Situación de partida.

B.4.5.2. Un caso interesante

Supongamos la situación de la figura B.15.

Para mezclar las ramas `client` y `master`, dejando de lado los cambios en la rama `server`, se utiliza la opción `--onto` del comando `git rebase`. Así, el comando

```
$ git rebase --onto master server client
```

indicará a Git que hay que

1. tomar los cambios de la rama `client`
2. determinar qué cambios hay que introducir en el ancestro común de `client` y `server`
3. aplicar esos cambios en la rama `master`

El resultado, que aparece en la figura B.16, es bastante interesante.

Por otra parte hay que tener claro que, al utilizar `rebase`, se borran los *commits* existentes y se crean nuevos *commits* que son parecidos, pero *no* exactamente iguales. Para evitar que se mezclen esos *commits* "antiguos" con los "nuevos" y evitar posibles errores graves, nunca se debe utilizar rebase para modificar commits que se encuentran registrados en un repositorio remoto.

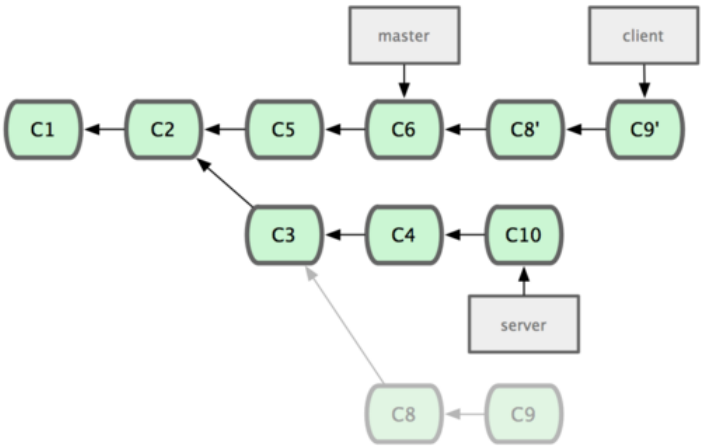

FIGURA B.16: Uso útil de `rebase`. Resultado obtenido.

Apéndice C

Organización del repositorio *GitHub*

C.1. Introducción

En este documento mostraremos cómo se encuentran distribuidos los ficheros que componen *CygnusCloud* en las diferentes ramas del repositorio de *GitHub*. Es posible acceder al mismo por medio de la dirección

https://github.com/lbarriosh/cygnus-cloud

A través del repositorio en *GitHub*, los usuarios podrán descargar el código, ver los diferentes *commits* realizados y los *issues* corregidos a lo largo del desarrollo de proyecto y también visualizar gráficas con las estadísticas del proyecto.

C.2. Descarga del contenido

Cualquier persona que lo desee puede descargar el contenido del repositorio a su propio ordenador para poder ejecutar y utilizar el código y cualquiera de los ficheros aportados en él, siempre y cuando cumpla la licencia asociada (véase el apéndice A).

La descarga del contenido del repositorio puede llevarse a cabo por medio de dos métodos diferentes:

- descargando un fichero comprimido `.zip` desde la página de *GitHub*.
- ejecutando el comando

```
git clone https://github.com/lbarriosh/cygnus-cloud.git
```

C.3. La rama `oldstable`

Esta rama contiene la versión de *CygnusCloud* que presentamos a la 7ª edición del Concurso Universitario de Software Libre. Los principales directorios de esta rama son los siguientes:

- `Arquitectura`. Este directorio contiene los diseños previos de arquitectura que ya han sido introducidos en la versión estable.
- `Certificados`. Incluye los ficheros de certificados utilizados en la versión estable.
- `Ficheros de configuración`. Incluye los ficheros de configuración utilizados por los distintos demonios de *CygnusCloud*.
- `src`. En este directorio se incluye el código de la primera versión estable de *CygnusCloud*.

C.4. La rama `master`

Esta rama contiene la última versión estable de *CygnusCloud*. El objetivo de esta rama es ir almacenando las versiones de *CygnusCloud* que ya han sido completamente terminadas y depuradas. Por lo tanto, esta rama nunca debe contener código a medio terminar o con posibles *bugs* contemplados para la versión que corresponda.

Los principales directorios que podemos encontrar en esta rama son:

- `Arquitectura`. Incluye tanto los diseños antiguos ya implementados en la última versión estable como los nuevos diseños que pueden *estar* o no implementados en la versión en desarrollo.
- `Certificados`. Incluye todos los certificados necesarios. En nuestro caso por lo general contendrá los mismos ficheros que la rama `oldstable`.
- `Documentación interna`. Incluye los documentos creados durante la familiarización con las herramientas utilizadas en la última versión estable y todos los documentos nuevos que hayan podido crearse para aprender a utilizar las herramientas empleadas en la versión en desarrollo.
- `Ficheros de configuración`. Incluye los ficheros de configuración de los diversos demonios de *CygnusCloud.*
- `Memoria`. En este directorio se incluye la memoria con los apartados que ya han sido revisados y pueden considerarse casi definitivos. Los miembros de *CygnusCloud* solo subirán un apartado a esta rama cuando este terminado y sea prácticamente definitivo.
- `src`. En este directorio se incluye todo el código de la última versión estable de *CygnusCloud*. Este código, permanece inalterado durante cada iteración, y sólo se actualiza cuando se completa una versión funcional del sistema. Dentro de este directorio podemos destacar dos subdirectorios:
  - `Infraestructura`. Incluye el código de toda la infraestructura, a excepción de la web.
  - `web`. Incluye el código de la aplicación web.

C.5. La rama `develop`

Esta rama esta destinada a mantener todo el código de la versión en desarrollo de *CygnusCloud*.

Debe garantizarse que todo el código subido a esta rama puede ser compilado y que no provoca ningún tipo de problema al resto de desarrolladores. Sin embargo, a diferencia de lo que ocurre en la rama `master`, el código que se sube en esta rama no tiene por qué estar completamente depurado.

En principio, en la rama `develop` existirán los mismos directorios que en la rama `master`, aunque estos podrán desaparecer o moverse sin previo aviso.

Apéndice D

BibTEX: guía de referencia

D.1. Configuración de BibTEX

El documento maestro ya está configurado para incluir las referencias. Para evitar conflictos (y, de paso, para tener ya clasificadas las referencias), cada capítulo de la memoria utilizará una base de datos bibliográfica distinta.

Esta base de datos no es más que un fichero de texto con extensión `.bib`, que podrá modificarse con cualquier editor.

D.2. Formato de la base de datos

D.2.1. Aspectos básicos

En el caso de los ficheros `.bib`,

1. Los espacios en blanco, tabuladores y saltos de línea no son más que delimitadores de palabras.
2. No se distingue entre mayúsculas y minúsculas.
3. Es posible utilizar caracteres acentuados (gracias a la configuración del documento maestro).

D.2.2. Registros y campos bibliotráficos

En toda base de datos bibliográfica, hay que distinguir entre:

- **Campos**. Son datos básicos aislados (como la fecha de publicación, el nombre de un autor,...).
- **Registros**. Es un conjunto de campos que describe una referencia bibliográfica.

En los ficheros `.bib`, los registros son de la forma

```
@TipoRegistro{clave,
    Campo1,
    Campo2,
    ...
    CampoN
}
```

donde:

- `TipoRegistro` identifica a un tipo concreto de registro (artículo, libro,...).

- La clave identifica al registro, y es siempre su primer elemento. En la misma base de datos *no* podrá haber dos registros con la misma clave, y tampoco deberían usarse bases de datos que compartan claves en el mismo documento. Por convenio, las claves serán de la forma

  ```
  <Tres iniciales apellido primer autor>Año
  ```

  Algunos ejemplos de clave son `Ull75`, `Knu85`. Si existen varias entradas con el mismo autor y el mismo año, se distinguirán añadiendo como sufijo una letra (`a` para la primera obra, `b` para la segunda, y así sucesivamente).

- Los campos pueden aparecer en cualquier orden. Para escribirlos, podemos usar uno de estos formatos:

  ```
  Campo = { Contenido }
  Campo = " Contenido "
  ```

  En cualquier caso,

  - Si no se especifica el nombre, el contenido se descartará.
  - Si un campo aparece más de una vez, se tomará el contenido correspondiente a su primera aparición.
  - Los delimitadores de contenido deben aparecer siempre que el contenido del campo no sea un número.

  Podemos distinguir tres tipos de campos:

  - Campos **obligatorios**. Deben aparecer obligatoriamente en determinados tipos de registro.
  - Campos **opcionales**. Si aparecen, el estilo los incluye.
  - Campos **ignorados**. El estilo los ignorará.

D.2.3. Tipos de registros bibliográficos

Se recogen en el cuadro D.1.

Registro	Descripción	Campos obligatorios	Campos opcionales
article	artículo publicado en una revista	author, title, journal, year	volumen, number, pages, month
book	libro normal	author/editor, title, publisher, year	volume/number, series, address, edition, month
booklet	folleto	title	author, howpublished, address, month, year
inbook	parte de un libro	author/editor, title (título del libro, no del capítulo), chapter/pages, publisher, year	volume/number, series, type, address, edition, month
incollection	parte de un libro con su propio título	author, title, booktitle, Publisher, year	crossref, editor, volume/number, type, chapter, pages, address, edition, month
inproceedings	actas	author, title, booktitle, year	crossref, editor, volume/number, series, pages, address, month, organization, publisher
manual	documentación técnica	title	author, organization, address, edition, month, year
masterthesis	proyecto fin de carrera	author, title, school, year	type, address, month
phdthesis	tesis doctoral	author, title, school, year	type, address, month
proceedings	libro de actas	title, year	booktitle, editor, volume/number, series, address, month, organization, publisher
techreport	informe técnico	author title, institution, year	type, number, address, month
unpublished	documento no publicado	author, title, note	month, year
misc	resto de casos	No tiene	author, title, howpublished, month, year

CUADRO D.1: Tipos de registros bibliográficos

Apéndice E

Sockets en *Python*

E.1. Introducción

Los *sockets* son un concepto abstracto con el que se designa al punto final de una conexión. Los programas utilizan *sockets* para comunicarse con otros programas, que pueden estar situados en computadoras distintas.

Un *socket* queda definido por la dirección IP de la máquina, el puerto en el que escucha, y el protocolo que utiliza. Los tipos y funciones necesarios para trabajar con *sockets* se encuentran en Python en el módulo `socket`.

E.2. Clasificación

Los *sockets* se clasifican en *sockets* de flujo (`socket.SOCK_STREAM`) o *sockets* de datagramas (`socket.SOCK_DGRAM`) dependiendo de si el servicio utiliza TCP, que es orientado a conexión y fiable, o UDP, respectivamente.

Los *sockets* también se pueden clasificar según la familia. Tenemos *sockets* UNIX (`socket.AF_UNIX`) que se crearon antes de la concepción de las redes y se basan en ficheros, *sockets* `socket.AF_INET` que son los que nos interesan, *sockets* `socket.AF_INET6` para IPv6, etc.

E.3. Familias de Sockets

Dependiendo del sistema y las opciones de construcción, existen varias familias de *sockets*.

Las direcciones de *sockets* pueden representarse como siguen:

- Las cadenas de caracteres son utilizadas para familias del tipo `AF_UNIX`.
- Un par de la forma `(host, port)`. Se utilizan para las familias de tipo `AF_INET` donde `host` es una cadena que representa un nombre de *host* en la notación de dominio de internet (como `daring.cwi.nl`) o una dirección IPv4 como (50.200.5.100), y `port` es un entero.
- En el caso de la familia de tipo `AF_INET6`, se utiliza una tupla de cuatro elementos `(host, port, flowinfo, scopeid)`, donde `flowInfo` y `scopeid` representan los miembros `sin6_flowinfo` y `sin6_scope_id` del registro `sockaddr_in6`. Estos parámetros pueden omitirse para compatibilidad con versiones anteriores. Sin embargo, la omisión de `ScopeId` puede causar problemas en la manipulación de ámbito de las direcciones IPv6.
- Los *sockets* de la familia `AF_NETLINK` se representan como pares `(pid, grupos)`.
- El soporte de TIPC en *Linux* solo está disponible bajo la familia `AF_TIPC`. TIPC es un protocolo abierto, sin IP y basado en red, diseñado para ser usado en ambientes con computación en *cluster*. Las direcciones son representadas por una tupla, y sus campos dependen del tipo de dirección. Generalmente suele tener el siguiente aspecto: `(addr_type, v1, v2, v3 ,[scope])` donde

  - `addr_type` es `TIPC_ADDR_NAMESEQ`, `TIPC_ADDR_NAME`, o `TIPC_ADDR_ID`.
    - Si `addr_type` es `TIPC_ADDR_NAME`, entonces `v1` es el tipo de servidor, `v2` es el identificador de puerto, y `v3` suele ser `0`.
    - Si `addr_type` es `TIPC_ADDR_NAMESEQ`, entonces `v1` es el tipo de servidor, `v2` es el número menor del puerto, y `v3` es el mayor número de puerto.
    - Si `addr_type` es `TIPC_ADDR_ID`, entonces `v1` es el nodo, `v2` es la referencia, y `v3` suele ser `0`.
  - `scope` es `TIPC_ZONE_SCOPE`, `TIPC_CLUSTER_SCOPE`, o `TIPC_NODE_SCOPE`.

- Algunas otras familias como `AF_BLUETOOTH` y `AF_PACKET` utilizan representaciones específicas.

E.4. Creación de un *socket*

Para crear un *socket* se utiliza el constructor `socket.socket()` que puede tomar como parámetros opcionales la familia, el tipo y el protocolo. Por defecto se utiliza la familia `AF_INET` y el tipo `SOCK_STREAM`.

E.4.1. *Socket* en el servidor

Lo primero que tenemos que hacer es crear un objeto *socket* para el servidor

```
socket_s = socket.socket()
```

Ahora tenemos que indicar en qué puerto se va a mantener a la escucha nuestro servidor utilizando el método `bind`. Para sockets IP, como es nuestro caso, el argumento de `bind` es una tupla que contiene el *host* y el puerto. El *host* se puede dejar vacío, indicando al método que puede utilizar cualquier nombre que esté disponible.

```
socket_s.bind(("localhost", 9999))
```

Por último, utilizamos `listen` para hacer que el *socket* acepte conexiones y *accept* para comenzar a escuchar. `listen` requiere un parámetro que indique el número de conexiones máximas que queremos aceptar. Evidentemente, este valor debe ser al menos `1`. `accept` se mantiene a la espera de conexiones entrantes, bloqueando la ejecución hasta que llegue un mensaje.

Cuando llega un mensaje, `accept` desbloquea la ejecución, devolviendo un objeto `socket` que representa la conexión del cliente y una tupla que contiene el *host* y el puerto de dicha conexión.

```
socket_s.listen(10)
socket_c, (host_c, puerto_c) = socket_s.accept()
```

Una vez que tenemos este objeto *socket* podemos comunicarnos con el cliente a su través, mediante los métodos `recv` y `send` (o `recvfrom` y `sendfrom` en UDP) que permiten recibir o enviar mensajes respectivamente. El método `send` toma como parámetros los datos a enviar, mientras que el método `recv` toma como parámetro el número máximo de bytes a aceptar.

```
recibido = socket_c.recv(1024)
print "Recibido: ", recibido
socket_c.send(recibido)
```

Una vez que hemos terminado de trabajar con el *socket*, lo cerramos con el método `close`.

E.4.2. *Socket* en el cliente

Crear un cliente es aún más sencillo. Solo tenemos que crear el objeto `socket`, utilizar el método `connect` para conectarnos al servidor y los métodos `send` y `recv` que vimos anteriormente para enviar y recibir mensajes. El argumento de `connect` es una tupla con el *host* y el puerto, exactamente igual que `bind`.

```
socket_c = socket.socket()
socket_c.connect(("localhost", 9999))
socket_c.send("hola")
```

E.5. Ejemplos

E.5.1. Ejemplo 1

En este ejemplo el cliente manda al servidor cualquier mensaje que escriba el usuario y el servidor no hace más que repetir el mensaje recibido. La ejecución termina cuando el usuario escribe `quit`.

Este sería el código del *script* servidor:

```
import socket
s = socket.socket()
s.bind(("localhost", 9999))
s.listen(1)
sc, addr = s.accept()
while True:
    recibido = sc.recv(1024)
    if recibido == "quit":
        break
    print "Recibido:", recibido
    sc.send(recibido)
print "adios"
sc.close()
s.close()
```

Y este el del *script* cliente:

```
import socket

s = socket.socket()
s.connect(("localhost", 9999))
while True:
       mensaje = raw_input("> ")
       s.send(mensaje)
       if mensaje == "quit":
           break
print "adios"
s.close()
```

E.5.2. Ejemplo 2

```
## python socket chat example
## licence: GPL v3
#server import socket import threading import time

SIZE = 4
soc = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```
soc.bind(('127.0.0.1',5432))
soc.listen(5)

class CThread(threading.Thread):
    def __init__(self,c):
        threading.Thread.__init__(self)
        self.conn = c
        self.stopIt=False

   def mrecv(self):
        data = self.conn.recv(SIZE)
        self.conn.send('OK')
        msg = self.conn.recv(int(data))
        return msg

   def run(self):
        while not self.stopIt:
            msg = self.mrecv()
            print 'recieved->  ',msg

def setConn(con1,con2):
    dict={}
    state = con1.recv(9)
    con2.recv(9)
   if state =='WILL RECV':
       dict['send'] = con1 # server will send data to reciever
       dict['recv'] = con2
   else:
       dict['recv'] = con1 # server will recieve data from sender
       dict['send'] = con2
   return dict

def msend(conn,msg):
    if len(msg)<=999 and len(msg)>0:
        conn.send(str(len(msg)))
        if conn.recv(2) == 'OK':
            conn.send(msg)
    else:
    conn.send(str(999))
       if conn.recv(2) == 'OK':
            conn.send(msg[:999])
            msend(conn,msg[1000:]) # calling recursive

(c1,a1) = soc.accept()
(c2,a2) = soc.accept()
dict = setConn(c1,c2)
thr = CThread(dict['recv'])
thr.start()
try:
    while 1:
    msend(dict['send'],raw_input())
except:
  print 'closing'

thr.stopIt=True
msend(dict['send'],'bye!!!')# for stoping the thread
```

```
thr.conn.close()
soc.close()

#client
import socket
import threading
SIZE =4
class client(threading.Thread):
    def __init__(self,c):
        threading.Thread.__init__(self)
        self.conn = c
        self.stopIt = False

   def mrecv(self):
        data = self.conn.recv(SIZE)
        self.conn.send('OK')
        return self.conn.recv(int(data))

   def run(self):
        while not self.stopIt:
            msg = self.mrecv()
            print 'recieved-> ',msg

soc1 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
soc1.connect(('127.0.0.1',5432)) soc1.send('WILL SEND')
# telling server we will send data from here
soc2 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
soc2.connect(('127.0.0.1',5432))
soc2.send('WILL RECV') # telling server we will recieve data from here

def msend(conn,msg):
if len(msg)<=999 and len(msg)>0:
        conn.send(str(len(msg)))
        if conn.recv(2) == 'OK':
            conn.send(msg)
    else:
        conn.send(str(999))
        if conn.recv(2) == 'OK':
            conn.send(msg[:999])
            msend(conn,msg[1000:]) # calling recursive

thr = client(soc2)
thr.start()
try:
    while 1:
        msend(soc1,raw_input())
except:
    print 'closing'

thr.stopIt=True
msend(soc1,'bye!!') # for stoping the thread
thr.conn.close()
soc1.close()
soc2.close()
```

Apéndice F

Creación de imágenes *Windows* en Xen

Los paquetes que imprescindiblemente tienen que estar instalados son:

- `xen-hypervisor-amd64` y sus dependencias.
- `blktap-utils` y `blktap-kms` para que la E/S a imágenes de disco sea eficiente.
- `bridge-utils` para poder configurar el *bridge* de Xen.

A continuación enumeraremos, paso por paso, todo lo necesario para llevar a cabo la creación de imagenes windows en Xen:

1. Elegimos la interfaz de red a la que queremos redirigir el tráfico de los adaptadores de red virtuales. Si sólo tenemos una, podemos saltar este paso, pero si tenemos dos (por ejemplo, *ethernet* y WiFi) tendremos que elegir *una* de ellas. Resulta recomendable utilizar la interfaz *ethernet* y desactivar la inalámbrica de la forma habitual.

2. Desactivamos `network-manager` (si no se ha hecho ya) para que podamos configurar la red sin problemas. Para desactivarlo, escribimos en un terminal

   ```
   sudo service network-manager stop
   ```

   Tras hacer esto, la dirección IP y el servidor DNS habrán desaparecido y no tendremos acceso a internet.

3. Modificamos el fichero `/etc/network/interfaces` para configurar la red. Suponiendo que queremos configurar *un bridge*, deberíamos añadir las líneas

   ```
   auto eth0
   allow-hotplug eth0
   iface eth0 inet dhcp
   auto xenbr0
   iface xenbr0 inet dhcp
       bridge_ports eth0
   ```

   Paraa consultar más información sobre el formato de este fichero, podemos usar las órdenes `man 5 interfaces` y `man 5 brige-utils-interfaces`.

4. Levantamos el *bridge* y la interfaz `eth0` mediante los comandos

   ```
   sudo ifup eth0
   ```

   ```
   sudo ifup xenbr0
   ```

5. Permitimos que todo tipo de tráfico pase a través del *bridge*. Para ello, usamos el comando

```
sudo iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
```

6. Parcheamos la instalación de `qemu` que trae Ubuntu por defecto: `qemu` espera encontrar los mapas de teclado en `/usr/share/qemu`, pero están en `/usr/share/qemu-linaro`. Lo arreglamos creando un enlace simbólico:

```
sudo ln -s /usr/share/qemu-linaro /usr/share/qemu
```

7. Creamos el fichero de configuración de domU que vamos a crear. Aunque suele recomendarse colocarlo en `/etc/xen`, resulta preferible almacenar las imágenes de disco y los ficheros de configuración en el mismo directorio (y lejos de `/etc/xen`). Esto facilita de manera considerable transferir máquinas virtuales ya configuradas entre distintas máquinas. En nuestro caso lo llamaremos `Windows.cfg`.

8. Creamos la imagen de disco. Para empezar, la crearemos como un fichero `.img`, si bien luego la comprimiremos en formato `vhd` (para ahorrar espacio). La orden a utilizar es

```
dd if=/dev/zero of=Windows.img bs=1 count=1 seek=30G
```

9. Fijamos el contenido del fichero `Windows.cfg`. Nosotros usaremos el siguiente:

```
# Habilitar virtualización completa
kernel = '/usr/lib/xen-4.1/boot/hvmloader'
builder = 'hvm'
# Configurar RAM, vcpus y shadow memory
memory = 3072
vcpus = 2
# Regla: 2 KB por MB de RAM más 1 MB por vcpu
shadow_memory = '8'
# Device model (procesa E/S del domU)
device_model = '/usr/lib/xen-4.1/bin/qemu-dm'
# Hostname
name = 'Windows'
# Almacenamiento: disco duro y CD-ROM
# ¡¡NO UTILIZAR ESPACIOS dentro de los strings!!
disk = ['tap2:tapdisk:aio:/media/Datos/xen-images/Windows/Windows.img,hda,w',
'phy:/dev/sr0,hdc:cdrom,r']
# Bridge a utilizar
vif = ['bridge=xenbr0']
# Secuencia de arranque: CD-ROM, disco duro
boot='dc'
# Habilitar ACPI
acpi = 1
apic = 1
# Configurar VNC
vnc=1
vncviewer=0
sdl=0
usbdevice='tablet'
# Tratamiento de eventos
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
```

10. Instalamos un cliente de escritorio remoto. Nosotros utilizaremos KRDC, aunque vale cualquiera.

11. Terminamos la instalación de `blktap-kms`. Para ello,

a) Configuramos DKMS. Así, cada vez que actualicemos el kernel, el módulo `blktap` no se dejará de cargar. Para ello, utilizamos el comando

```
sudo dkms autoinstall -k $(uname -r)
```

b) Cargamos el módulo `blktap`. Deberá hacerse esto en cada arranque.

```
modprobe blktap
```

12. Arrancamos la máquina virtual de la forma habitual, y arrancamos el cliente de escritorio remoto. Nos conectamos a `localhost` (no es necesario introducir ninguna contraseña). Si lo hemos hecho todo bien, veremos cómo aparece el BIOS emulado y, después, el instalador de Windows 7.

    A la hora de finalizar la instalación,

    - el nombre del equipo deberá ser de la forma

      ```
      Xen-Win[XP|Vista|7][Pro|Ent]-[x86_32|x86_64]
      ```

    - creamos el usuario `VM-Admin`, con contraseña `CygnusCloud`.
    - habilitamos las actualizaciones automáticas.

    Observación: en cada reinicio, el domU se destruirá, y tendremos que volver a crearlo. Este comportamiento es *normal*, y persistirá hasta que instalemos los drivers de paravirtualización.

13. Instalamos los drivers GPL para habilitar la paravirtualización. Los instaladores que necesitamos son:

Sistema operativo	Arquitectura	Fichero (de meadowcourt.org/downloads)
Windows XP	x86_32	`gplpv_XP_0.11.0.357.msi`
Windows Vista Windows 7 Windows Server 2008 Windows 8	x86_32	`gplpv_Vista2008x32_0.11.0.357.msi`
Windows Vista Windows 7 Windows Server 2008 Windows 8	x86_64	`gplpv_Vista2008x64_0.11.0.357.msi`

    Hay varias alternativas para pasar el instalador a la máquina virtual. Resulta recomendable copiarlos en algún dispositivo extraible y enchufarlo al domU añadiendo lo que corresponda a la línea `disk=` del fichero de configuración. En nuestro caso, lo que demos añadir es:

    ```
    'phy:/dev/sdc,hdb,w'
    ```

    Para saber qué dispositivo es el dispositivo extraible utilizado, podemos montarlo con el *shell* y después utilizar el comando `mount`. En nuestro caso la salida debe ser algo como esto

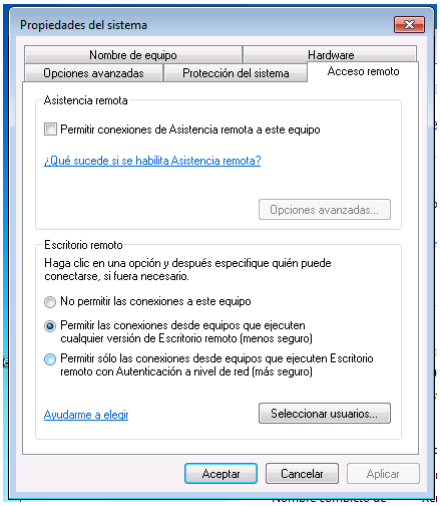

FIGURA F.1: Configuración de RDP en Windows 7 y Vista

```
/dev/sda5 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)

tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)

none on /run/shm type tmpfs (rw,nosuid,nodev)
/dev/sda6 on /home type ext4 (rw)
/dev/sda3 on /media/Datos type fuseblk
(rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)
/dev/sda2 on /media/Windows type fuseblk
(rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)
xenfs on /proc/xen type xenfs (rw)
/dev/sdc1 on /media/Luis type fuseblk
   (rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)
```

El 1 que aparece es el número de partición: el dispositivo extraible sólo tiene una, por lo que es la primera.

En el siguiente reinicio, notaremos que nuestra máquina virtual es algo más rápida.

14. Configuramos RDP. Este protocolo de escritorio remoto es el que mejor funciona con máquinas *Windows*. Los pasos a seguir (en Windows Vista/7) son:

    a) Hacemos clic derecho sobre Equipo ▷ Propiedades.

    b) Hacemos clic sobre Configuración Avanzada del Sistema, y luego en la pestaña Acceso remoto. Dejamos las cosas como aparece en la figura F.1.

    c) Cerramos sesión e intentamos conectarnos al domU usando RDP. Como nombre de host, usamos el que hemos especificado en la instalación (`Xen-Win7Pro-x64` en nuestro caso).

    d) Desactivamos VNC. Para ello, cambiamos la línea `vnc=1` del fichero de configuración por `vnc=0`.

> **Nota**: para apagar el domU, es necesario utilizar el comando `shutdown /n`.

15. Quitamos todos los dispositivos de almacenamiento salvo el disco duro. Para ello, modificamos la línea `disk=` del fichero de configuración.
16. Configuramos el arranque para que sólo se utilice el disco duro. Para ello, modificamos la línea `boot='dc'` del fichero de configuración para que sea `boot='c'`.
17. Comprimimos la imagen de disco y borramos la antigua (el fichero `.img`). Para ello, utilizamos la orden

    ```
    qemu-img convert Windows.img -O qcow2 Windows.qcow2
    ```

18. Modificamos el fichero de configuración para que utilice la nueva imagen. Para ello, sustituimos

    ```
    'tap2:tapdisk:aio:/media/Datos/xen-images/Windows/Windows.img,hda,w'
    ```

    por

    ```
    'tap2:tapdisk:qcow:/media/Datos/xen-images/Windows/Windows.qcow,hda,w'
    ```

Una vez hecho esto, ya estaría todo listo para utilizar nuestra imagen windows en Xen.

Apéndice G

Configuración de KVM

Los pasos a seguir para configurar de forma correcta KVM son los siguientes:

1. Nos aseguraremos de que tenemos una CPU con extensiones de virtualización. Si no es el caso, KVM no funcionará. Para ello, usamos la orden

   ```
   egrep '(vmx|svm)' --color=always /proc/cpuinfo flags
   ```

   Un posible ejemplo de salida para esta orden sería la siguiente:

   ```
   luis@luis-desktop:~$ egrep '(vmx|svm)' --color=always
   /proc/cpuinfo
   flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
   pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht
   tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_-
   good nopl aperfmperf pni dtes64 monitor ds_cpl vmx smx est tm2
   ssse3 cx16 xtpr pdcm sse4_1 xsave lahf_lm dtherm tpr_shadow vnmi
   flexpriority
   ```

   En ella aparecerán tantos bloques como *cores* tenga integrados nuestra CPU. En el caso de CPUs Intel, el flag es `vmx`; en el caso de CPUs AMD, el flag es `svm`.

2. Instalamos los paquetes básicos con la orden

   ```
   sudo apt-get install ubuntu-virt-server python-vm-builder kvm-ipxe
   ```

3. Añadimos a *root* a los grupos `libvirt` y `kvm`. También añadimos a root nuestro propio usuario, para no tener que estar usando el comando sudo todo el rato. Para ello, utilizamos el comando `adduser` de la siguiente forma:

   ```
   adduser <nombre de usuario> <grupo>
   ```

   Un posible ejemplo sobre esto sería el siguiente:

   ```
   sudo adduser root libvirtd
   sudo adduser luis libvirtd
   sudo adduser root kvm
   sudo adduser luis kvm
   ```

   Reiniciamos sesión para que los cambios tengan efecto.

4. Comprobamos que KVM funciona, mediante el comando

   ```
   virsh -c qemu:///system list
   ```

La salida será de la forma:

```
luis@luis-desktop:~$ virsh -c qemu:///system
list
Id Name State
----------------------------------
```

Lógicamente, todavía no hemos creado ninguna máquina virtual, por lo que la lista es vacía.

5. Configuramos el bridge, editando el fichero `/etc/network/interfaces` para que quede de esta forma:

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
auto kvmbr0
iface kvmbr0 inet dhcp
    bridge_ports eth0
```

6. Detenemos network-manager y reiniciamos el demonio de red.

```
sudo service network-manager stop
sudo /etc/init.d/networking restart
```

7. Activamos las interfaces que acabamos de definir

```
sudo ifconfig
```

8. Reiniciamos. Esto es necesario para evitar errores de permisos en `/dev/kvm`.
9. Modificamos los ficheros `.xml` del repositorio según el tipo de dominio que vayamos a crear (Windows o GNU/Linux).
10. Arrancamos la máquina virtual

```
virsh define <ruta del fichero .xml>
virsh start <nombre de la VM>
```

11. Iniciamos un cliente de escritorio remoto, conectándonos a `localhost`. Por defecto, no será necesario introducir un nombre de usuario ni una contraseña.

Apéndice H

Configuración de una red virtual en modo NAT

H.1. ¿Qué queremos hacer?

Al usar NAT, expondremos al exterior una única dirección IP para el servidor de máquinas virtuales. El tráfico de las distintas máquinas virtuales se redirigirá a puertos asociados a esta IP.

Para configurar NAT, hacen falta tres cosas:

- una puerta de enlace (*gateway*), que conectará nuestras máquinas virtuales al exterior.
- una receta de iptables, que determinará cómo redirigir el tráfico destinado a la IP del servidor a las distintas máquinas virtuales.
- un servidor DHCP, que asignará IPs a las máquinas virtuales y les proporcionará un servidor DNS.

Nosotros lo configuraremos todo en ese orden.

Antes de continuar, debemos deshabilitar network-manager para poder modificar con total libertad la configuración de la red.

H.2. Creación del bridge

Para crear y configurar el bridge, utilizaremos el comando `brctl`.

1. Creamos el *bridge*

```
brctl addbr kvmbr0
```

2. Desactivamos STP (*Spanning Tree Protocol*). En nuestra red virtual no habrá ciclos, por lo que no debemos preocuparnos por ellos.

```
brctl stp kvmbr0 off
```

3. Configuramos el *forward delay* del bridge a 0 segundos.

```
brctl setfd kvmbr0 0
```

Una vez creado el bridge, debemos registrarlo como interfaz de red. Arbitrariamente, le asignaremos la dirección IP `192.168.77.1`.

La primera máquina tendrá la IP `192.168.77.2`, la segunda, `192.168.77.3`, y así sucesivamente, por lo que la máscara de red debe ser `255.255.255.0`.

La orden que nos permite configurar todo esto es

```
ifconfig kvmbr0 192.168.77.1 netmask 255.255.255.0 up
```

Antes de hacer esto, debemos comprobar si el *bridge* existe o no. Para ello usamos:

```
brctl show | grep "^kvmbr0" > /dev/null 2> /dev/null
```

Con saber si esta orden falla o no, será suficiente. La salida del grep no nos interesa.

H.3. Activar el encaminamiento IP

El siguiente paso es activar el encaminamiento (o *forwarding*) IP, desactivado por defecto en todas las distribuciones de escritorio.
El comando que debemos utilizar es

```
echo 1 | dd of=/proc/sys/net/ipv4/ip_forward > /dev/null
```

De manera predeterminada, este ajuste se pierde tras reiniciar. Si queremos hacerlo permanente, debemos descomentar la siguiente línea

```
#net.ipv4.ip_forward=1
```

del fichero `/etc/sysctl.conf`.

H.4. Configuración de `iptables`

Antes de nada, resulta muy recomendable guardar nuestra receta actual para evitar problemas. Podemos hacerlo mediante el comando

```
iptables-save > ruta_del_fichero
```

Para recuperar los ajustes, usamos el comando

```
iptables-restore < ruta_del_fichero
```

Ejecutamos estas órdenes[1]:

```
iptables -t nat -A POSTROUTING -s 192.168.77.0/255.255.255.0 -j MASQUERADE
iptables -t filter -A INPUT -i kvmbr0 -p tcp -m tcp --dport 67 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p udp -m udp --dport 67 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p tcp -m tcp --dport 53 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p udp -m udp --dport 53 -j ACCEPT
iptables -t filter -A FORWARD -i kvmbr0 -o kvmbr0 -j ACCEPT
iptables -t filter -A FORWARD -s 192.168.77.0/255.255.255.0 -i kvmbr0 -j ACCEPT
iptables -t filter -A FORWARD -d 192.168.77.0/255.255.255.0 -o kvmbr0 -m state
--state RELATED,ESTABLISHED -j ACCEPT
iptables -t filter -A FORWARD -o kvmbr0 -j REJECT --reject-with
icmp-port-unreachable
iptables -t filter -A FORWARD -i kvmbr0 -j REJECT --reject-with
icmp-port-unreachable
```

Para poder entender estas líneas, debemos tener en cuenta que:

- En el caso de la tabla nat, `PREROUTING` indica que los datagramas entrantes se alterarán, `POSTROUTING` indica que los datagramas salientes se alterarán, y `OUTPUT` indica que los datagramas que se generan en esta máquina también se alterarán.

[1] De forma predeterminada, `iptables` trabaja con la tabla `filter`. Por eso, el argumento `-t filter` se puede omitir.

- ACCEPT indica que hay que dejar pasar el datagrama.
- MASQUERADE es la dirección IP del servidor de máquinas virtuales.
- En el caso de la tabla filter, INPUT hace referencia a paquetes destinados a esta máquina, FORWARD hace referencia a paquetes que se enrutarán a través de esta máquina y OUTPUT hace referencia a paquetes generados en esta máquina.
- Las primera línea -A INPUT tiene el siguiente significado

  - si el datagrama va destinado a la interfaz kvmbr0, es decir, a alguna máquina virtual,
  - lleva datos del protocolo TCP, y
  - está destinado al puerto TCP 67

  lo aceptamos. El resto tiene un significado similar.

  Los puertos 67 y 53 son los utilizados por los servicios DHCP y DNS.
- Como podemos ver, no hemos conectado aun el *bridge* a nada. Para permitir a los datagramas generados por las máquinas virtuales pasar a través del bridge, debemos utilizar la siguiente instrucción *que se indica en la siguiente linea.*
- Las siguiente línea redirige los paquetes procedentes de la red (es decir, del exterior) al bridge.
- La siguiente línea redirige los paquetes procedentes del *bridge* (es decir, generados por las máquinas virtuales) al exterior. RELATED y ESTABLISHED hacen referencia al estado de la conexión. Si queremos obtener más detalles, tendremos que consultar el fichero man iptables.
- Las dos últimas líneas descartan los datagramas asociados a errores ICMP.

Para comprobar que la configuración es correcta, usamos los comandos

```
iptables -t nat -L -n
```

```
iptables -t filter -L -n
```

H.5. Configuración de dnsmasq

Para configurar el servidor DNS, esjecutamos esta orden:

```
dnsmasq --strict-order --except-interface=lo
--except-interface=eth0 --except-interface=vnet0
--interface=kvmbr0 --listen-address=192.168.77.1
--bind-interfaces --dhcp-range=192.168.77.2,192.168.77.254
--conf-file="" –dhcp-leasefile=/home/luis/kvmbr0.leases
--dhcp-no-override
```

La línea resaltada es la que nos interesa: en ese fichero, figurarán todas las asignaciones de IPs que realiza el servidor DHCP.

H.6. Redirección del tráfico de un puerto al servidor VNC de una máquina virtual

Por ahora, el servidor VNC corre a cargo de KVM y QEMU, por lo que reside en el servidor de máquinas virtuales.

Esto significa que, por ahora, no es necesario redirigir el tráfico de un puerto del servidor a una máquina virtual. No obstante, podemos forzar la redirección explícitamente utilizando las siguientes lineas.

```
iptables -t nat -I PREROUTING -p <tcp|udp> --dport <puerto
servidor> -j DNAT --to-destination <IP VM>:<Puerto VM> iptables
-A FORWARD -i eth0 -o kvmbr0 -p <tcp|udp> --dport <Puerto VM> -j
ACCEPT
```

El primer comando nos permite garantizar que no habrá problemas al crear y utilizar conexiones en las máquinas virtuales. El segundo es el que redirige todo el tráfico de cierto puerto del servidor a cierto puerto de la máquina virtual.

Bibliografía

[1] INE, "Alumnos matriculados por universidad (2011-2012)." Disponible en http://www.ine.es/jaxi/tabla.do?type=pcaxis&path=/t13/p405/a2010-2011/l0/&file=02011.px.

[2] N. I. o. S. Technology, "The NIST definition of Cloud Computing," *Special Publication 800-145, National Institute of Standards and Technology*, July 2011. Disponible en http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[3] P. McFedries, "The Cloud Is The Computer." Disponible en http://spectrum.ieee.org/computing/hardware/the-cloud-is-the-computer, Aug. 2008.

[4] Intrinsic Technology, "HVD: the cloud's silver lining." Disponible en http://www.intrinsictechnology.co.uk/FileUploads/HVD_Whitepaper.pdf, Aug. 2012.

[5] T. Chou, *Introduction to Cloud Computing: business & technology*. Active Book Press, 2nd ed., 2011.

[6] J. Hurwitz, R. Bloor, M. Kaufman, and F. Halper, *Cloud computing for dummies*. John Wiley & Sons, Nov. 2009.

[7] F. Gens, "Defining "Cloud services" and "Cloud computing"." Disponible en http://blogs.idc.com/ie/?p=190, Sept. 2008.

[8] A. C. Murray, "There's No Such Thing As A Private Cloud." Disponible en http://www.informationweek.com/cloud-computing/theres-no-such-thing-as-a-private-cloud/229207922, June 2010.

[9] G. Haff, "Just don't call them private clouds." Disponible en http://news.cnet.com/8301-13556_3-10150841-61.html, Jan. 2009.

[10] A. Stevens, "When hybrid clouds are a mixed blessing." Disponible en http://www.theregister.co.uk/2011/06/29/hybrid_cloud, June 2012.

[11] R. Johnson, "Cloud computing is a trap, warns GNU founder Richard Stallman." Disponible en http://www.guardian.co.uk/technology/2008/sep/29/cloud.computing.richard.stallman, 2008.

[12] T. Deshane, Z. Shepherd, Z. Shepherd, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative Comparison of Xen and KVM," in *Xen Summit proceedings*, jun 2008.

[13] Twisted Matrix Labs, *Twisted 12.3.0 core documentation*, 12.3.0 ed., Dec. 2012. Disponible en http://twistedmatrix.com/documents/current/core/howto/index.html.

[14] A. Fettig, *Twisted network programming essentials*. O'Reilly Media, 2005.

[15] M. Di Pierro, *web2py, Complete Reference Manual.*, 5th ed., 2013. Disponible en http://web2py.com/book.

[16] M. Di Pierro, *Access Control*, 5th ed., 2013.

[17] OWASP, "OWASP: The Open Application Security Project," 2011. Disponible en https://www.owasp.org/index.php/Category:How_To.

[18] J. W. Plattner and L. W. Herron, "Simulation: Its Use in Employee Selection and Training," *American Management Association Bulletin*, vol. 20, 1962.

[19] M. Carnoy, "ITC in Education: Possibilities and Challenges," in *Apertura del curso académico*, Universitat Oberta de Catalunya, Oct. 2004. Disponible en www.uoc.edu/inaugural04/eng/carnoy1004.pdf.

[20] F. Ginés, J. M. Mirones, E. Sánchez, D. Soria, R. Ruiz, T. Hortalá, and J. M. Mendías, "Laboratorios Docentes de la Facultad de Informática de la UCM: un modelo para la gestión integrada de aulas informáticas," in *Primera Jornada Campus Virtual UCM*, pp. 146–153, Universidad Complutense de Madrid, Nov. 2004. Disponible en eprints.ucm.es/5501/1/I_JORNADA_CAMPUS.pdf.

[21] "ENERGY STAR Program Requirements for Computers," tech. rep., US Environmental Protection Agency, 2009. Disponible en http://www.energystar.gov/index.cfm?c=computers.pr_crit_computers.

[22] "Memoria de la Facultad de Informática, Curso 2009/2010." Disponible en http://www.fdi.ucm.es/Futuros_Alumnos/memoria_curso_2009-2010.pdf.

[23] Wikipedia, "Original equipment manufacturer." Disponible en http://en.wikipedia.org/wiki/Original_equipment_manufacturer. Consultado el 25/10/2012.

[24] P. S. Ryan, S. Falvey, and R. Merchant, "Regulation of the cloud in India," *Journal of Internet Law*, Oct. 2011. Disponible en http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1941494.

[25] E. Mills, "Cloud computing security forecast: clear skies." Disponible en http://news.cnet.com/8301-1009_3-10150569-83.html, Jan. 2009.

[26] M. Miller, *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. QUE, 2008.

[27] C. Evitech, "Historia antigua del tratamiento del agua potable." Also available at http://blog.condorchem.com/historia-antigua-del-tratamiento-del-agua-potable/, Jan. 2010.

[28] Salesforce, "¿Qué es Cloud Computing?." Disponible en http://www.salesforce.com/es/cloudcomputing/.

[29] G. Gilder, "The information factories," *Wired*, Oct. 2006. Disponible en http://www.wired.com/wired/archive/14.10/cloudware.html.

[30] B. Sosinsky, *Cloud Computing Bible*. John Wiley & Sons, Jan. 2011.

[31] Wikipedia, "Cloud Computing." Disponible en https://en.wikipedia.org/wiki/Cloud_computing. Consultado el 25/11/12.

[32] A. Escoms, I. Espinar, and E. Rodrigo, "Constructor: plataforma como servicio en la nube para startups," Master's thesis, Facultad de Informática, 2012. Disponible en http://eprints.ucm.es/16089/1/Memoria.pdf.

[33] A. Megía, A. Molinera, and J. A. Rueda, "RSA@Cloud: criptoanálisis eficiente en la nube," Master's thesis, Facultad de Informática, 2011.

[34] C. D. Graziano, "A performance analysis of Xen and KVM hypervisors," Master's thesis, Iowa State University, 2011. Disponible en http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3243&context=etd.

[35] M. Larabel, "Ubuntu 12.10: Linux KVM vs. Xen Virtualization Preview." Disponible en http://www.phoronix.com/scan.php?page=article&item=ubuntu1210_xenkvm_preview&num=1.

[36] C. Takemura and L. S. Crawford, *The Book of Xen: A Practical Guide for the System Administrator*. William Pollock, oct 2009.

[37] Wikipedia, "Virtualization." Disponible en http://en.wikipedia.org/wiki/Virtualization.

[38] Wikipedia, "Hardware Virtualization." Disponible en http://en.wikipedia.org/wiki/Hardware_virtualization.

[39] Wikipedia, "Virtual machine." Disponible en http://en.wikipedia.org/wiki/Virtual_machine.

[40] Wikipedia, "Full virtualization." Disponible en http://en.wikipedia.org/wiki/Full_virtualization.

[41] Wikipedia, "Paravirtualization." Disponible en http://en.wikipedia.org/wiki/Paravirtualization.

[42] Wikipedia, "Hypervisor." Disponible en http://en.wikipedia.org/wiki/Hypervisor.

[43] J. Martin and H. Ichikawa, "noVNC and websockify documentation." Disponible en http://kanaka.github.com/noVNC/.

[44] D. Peticolas, "An introduction to Twisted." Disponible en http://krondo.com/?page_id=1327. Consultado en enero de 2013.

[45] "KVM Official documentation." Disponible en http://www.linux-kvm.org/page/Documents.

[46] "Libvirt Language Bindings." Disponible en http://libvirt.org/bindings.html.

[47] "Libvirt: supported hypervisors." Disponible en http://libvirt.org/drivers.html#hypervisor.

[48] M. Lutz and D. Ascher, *Learning Python*. O'Reilly Media, 4 ed., 2011.

[49] M. Lutz and D. Ascher, *Programming Python*. O'Reilly Media, 4 ed., 2011.

[50] M. Pilgrim, *Dive into Python*. APress, 2004. Disponible en http://www.diveintopython.net/.

[51] G. van Rossum, *Python tutorial*, 2005.

[52] M. Pilgrim, *HTML5: Up and Running*. O'Reilly Media, aug 2010. Disponible en http://diveintohtml5.info/.

[53] w3schools.com, "w3schools HTML4, HTML5, CSS and JavaScript tutorials." Disponible en http://www.w3schools.com/.

[54] N. Inc., "SUSE Linux Enterprise Server Virtualization With Xen." Disponible en http://doc.opensuse.org/products/draft/SLES/SLES-xen_sd_draft/book.xen.html, jun 2012.

[55] N. Inc., "openSUSE 12.3 Virtualization with KVM." Disponible en http://doc.opensuse.org/products/draft/SLES/SLES-xen_sd_draft/book.xen.html, mar 2013.

[56] U. Project, "Xen." Disponible en https://help.ubuntu.com/community/Xen.

[57] U. Project, "KVM." Disponible en https://help.ubuntu.com/community/KVM.

[58] Disponible en http://virt-manager.org/documentation/.

[59] Disponible en https://kb.askmonty.org/en/.

[60] Disponible en http://docs.python.org/2/install/#inst-new-standard.

[61] IBM, "IBM Linux Blueprint: Quick Start Guide for installing and running KVM." Disponible en http://pic.dhe.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaai/kvminstall/liaaikvminstallstart.htm, mar 2013.

[62] P. S. L. Documentation, "socket: Low-level networking interface."

[63] G. McMillan, "Socket Programming HOWTO." Disponible en http://docs.python.org/3.1/howto/sockets.html.

[64] A. Shrivastava, "A simple Python socket chat example." Disponible en http://ankurs.com/2009/07/a-simple-python-socket-chat-example/.

[65] S. Chacon, *Pro Git*. Apress, aug 2009.